Speech and Language Algorithms Group

Deep Neural Networks With Applications to Speech and Language Processing

Tara N. Sainath
IBM T.J. Watson Research Center
February 3, 2014

Acknowledgements

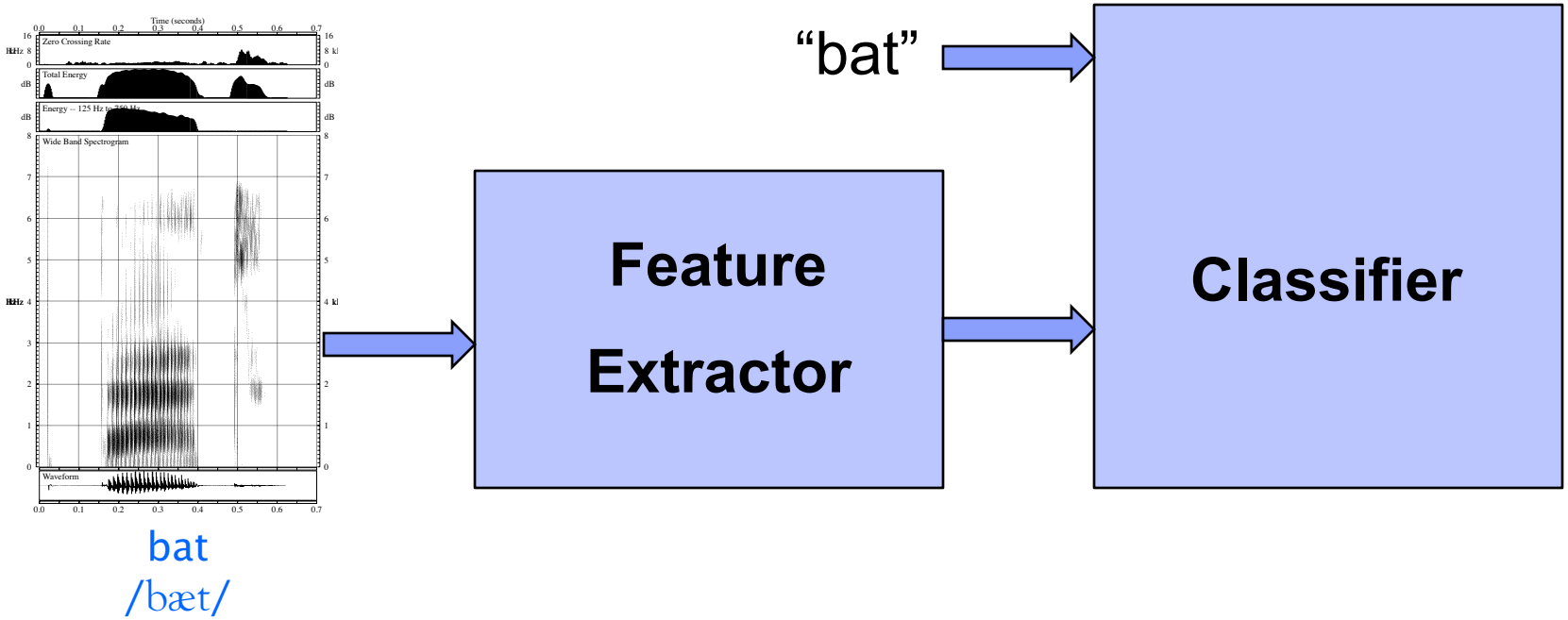
- IBM Speech Dept: Brian Kingsbury, George Saon, Hagen Soltau, Bhuvana Ramabhadran, Michael Picheny
- IBM Math Dept: Vikas Sindhwani, Lior Horesh, Sasha Aravkin, I-shin Chung, John Gunnels
- University of Toronto: Abdel-rahman Mohamed, George Dahl, Geoff Hinton

Outline

- Why Neural Networks?
- Training Neural Networks
- Making Deep Neural Network (DNNs) Successful For Speech Recognition
  - Pre-training
  - GPUs
- Research Directions and Challenges

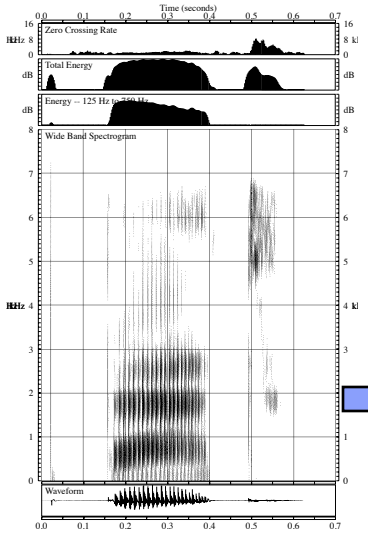
Pattern Recognition System

- The goal of any pattern recognition system is to
  - determine an appropriate feature representation
  - classify these features effectively

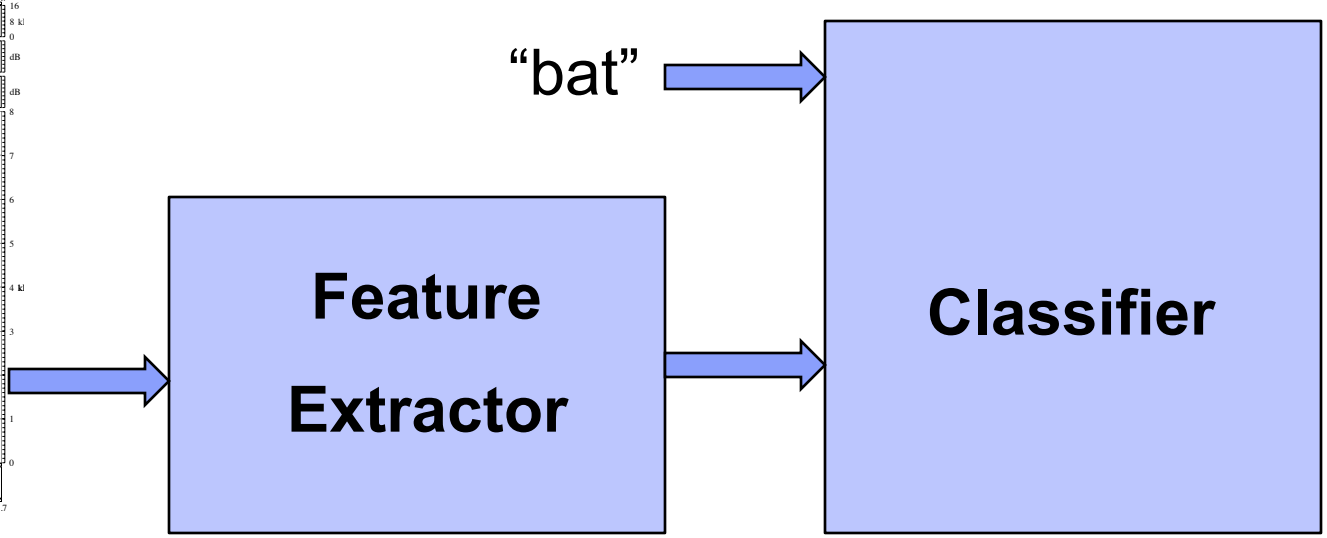


Example from Speech Recognition

- For example, in speech recognition, we first create features by hand
- Then we build a discriminative classifier (Gaussian Mixture Model) to distinguish between classes
- Features are not directly designed for classification objective

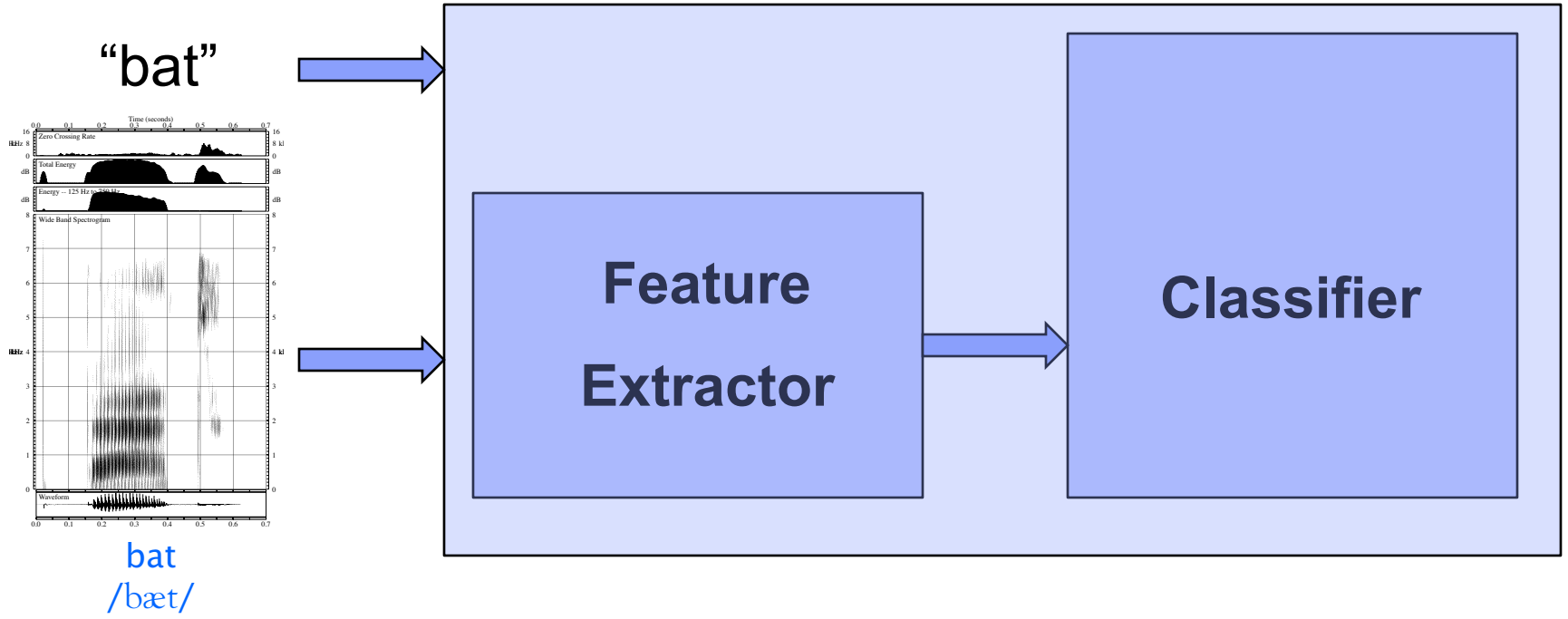


bat
/bæt/



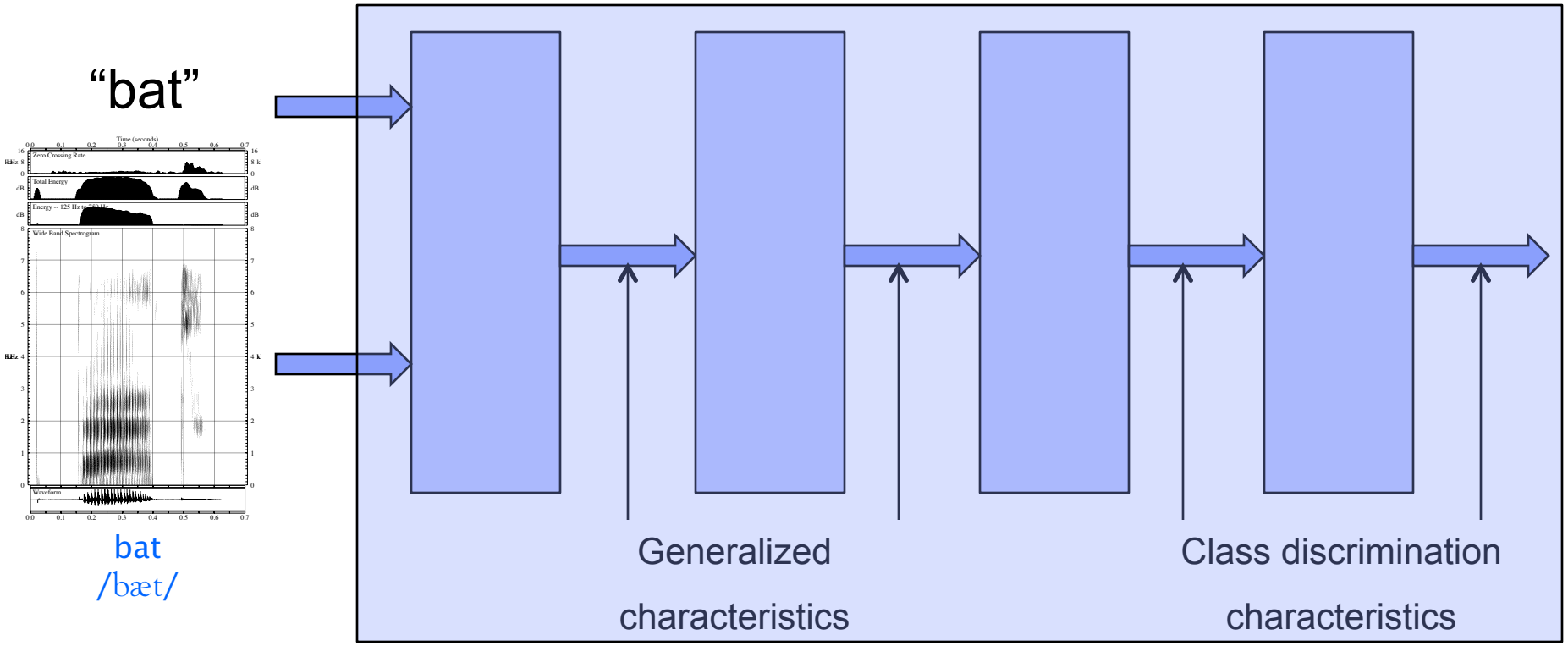
End-to-End Recognition System

- Black box which takes simple features + labels does the feature extraction **jointly** with the classification
- Features are trained to the classification objective
- Big non-linear system trained to map from simple features to labels



Intuition Behind Deep Neural Networks

- Each block produces a higher level feature representation and better classifier than its input
- By combining simple building blocks, we can design more and more complex, non-linear systems

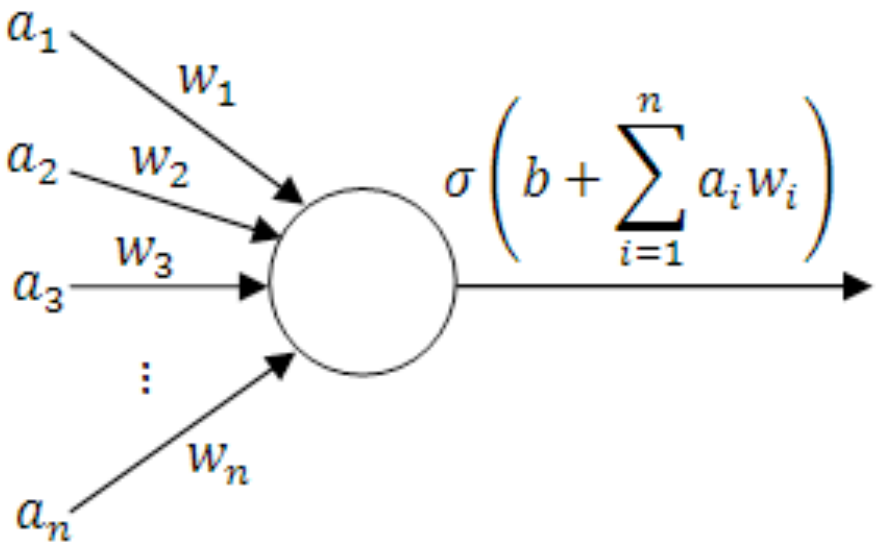


Representation at Each Stage

- Within each building block, we want the following properties:
  - Create a higher level representation of input
  - Better separate input into classes
  - Can be combined with previous layers
  - Can be trained jointly with other layers
- Using a mathematical model of a biological neuron is an appropriate choice

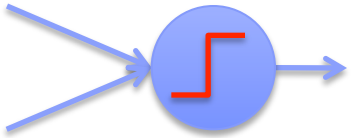
Neurons

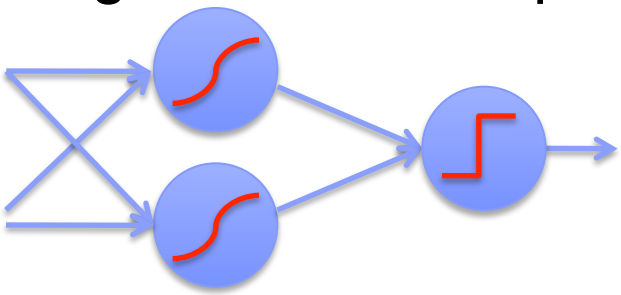
- A neuron takes a weighted sum of inputs ***a*** and feeds the result through an activation function ***σ***
- Output of the activation function produces decision boundary which can be used in classification

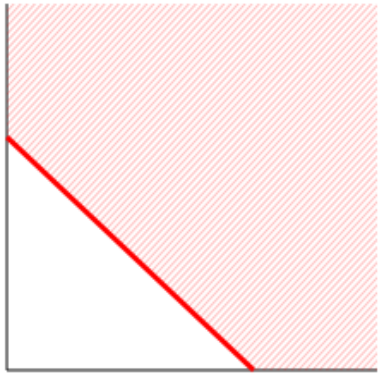


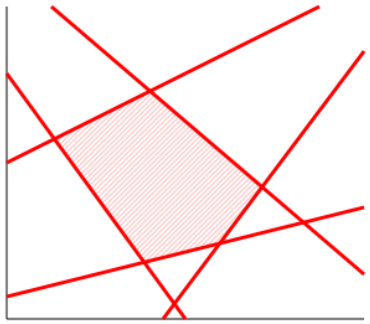
Combining Neurons

- Each neuron splits the feature space with a hyperplane
- 1-layer of trainable weights cannot handle XOR
- 2-layers of trainable weights gives a convex polygon region



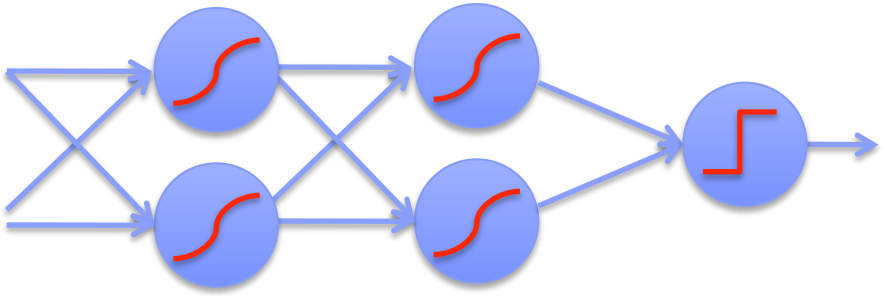


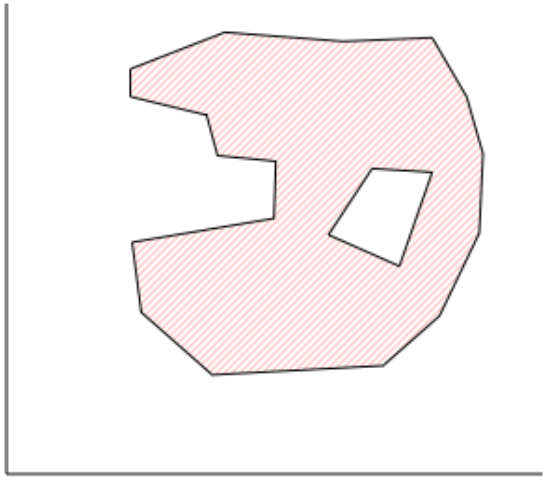




Combining Neurons

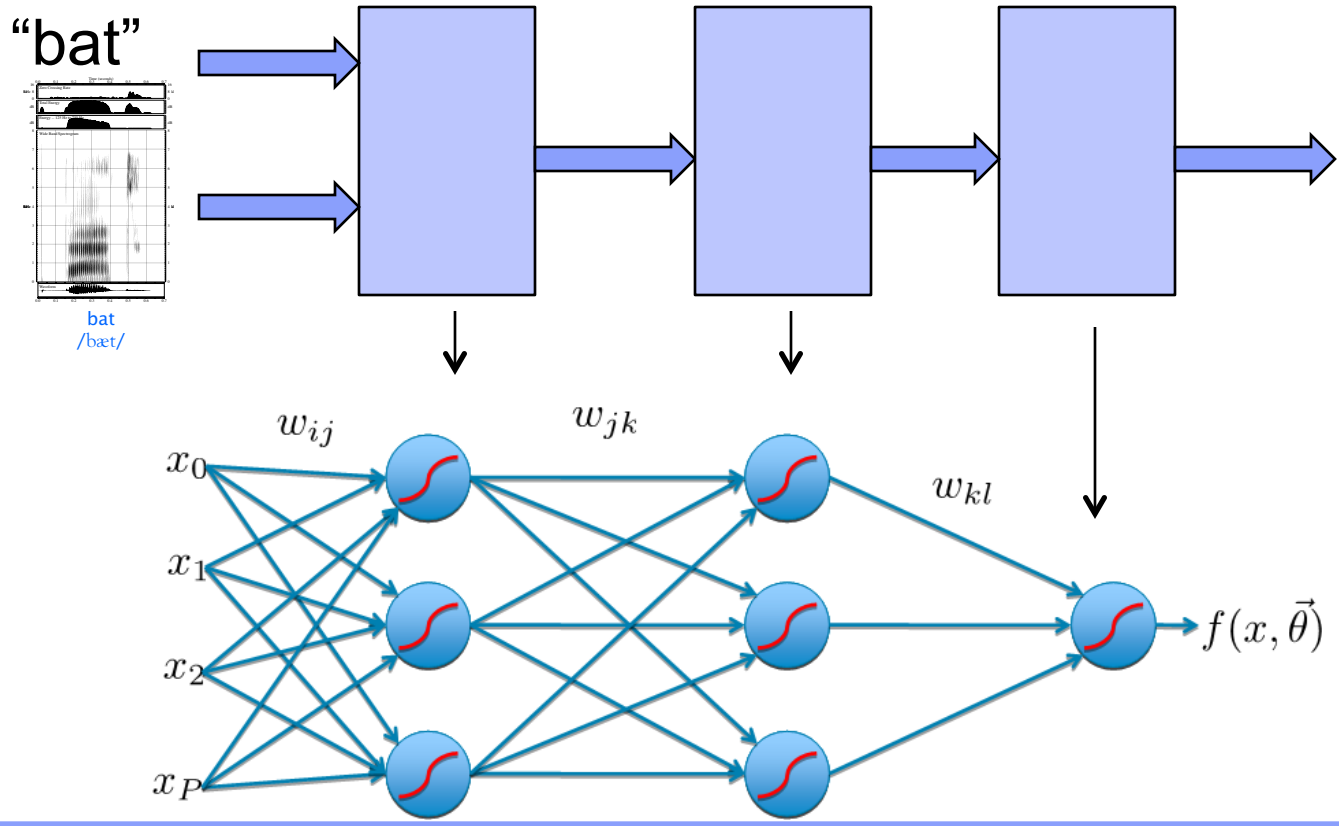
- 3 layers of trainable weights gives a composition of polygons: convex regions
- More layers can handle more complicated spaces – but require more parameters





Multi-Layer Neural Network

- Each simple building block is a connection of neurons which produces a higher-order, more complex representation of the input
- Neurons in one layer are connected to neurons in the next layer



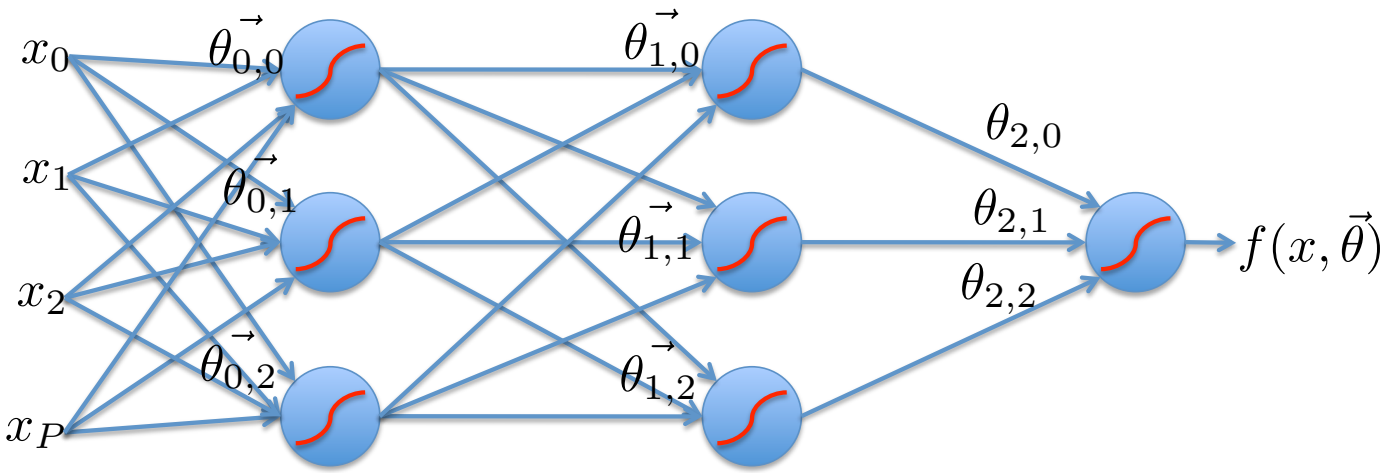
Outline

- Why Neural Networks?
- Training Neural Networks*
- Making Deep Neural Network (DNNs) Successful For Speech Recognition
  - Pre-training
  - GPUs
- Research Directions and Challenges

*** Thanks to Andrew Rosenberg for Backpropagation Figures**

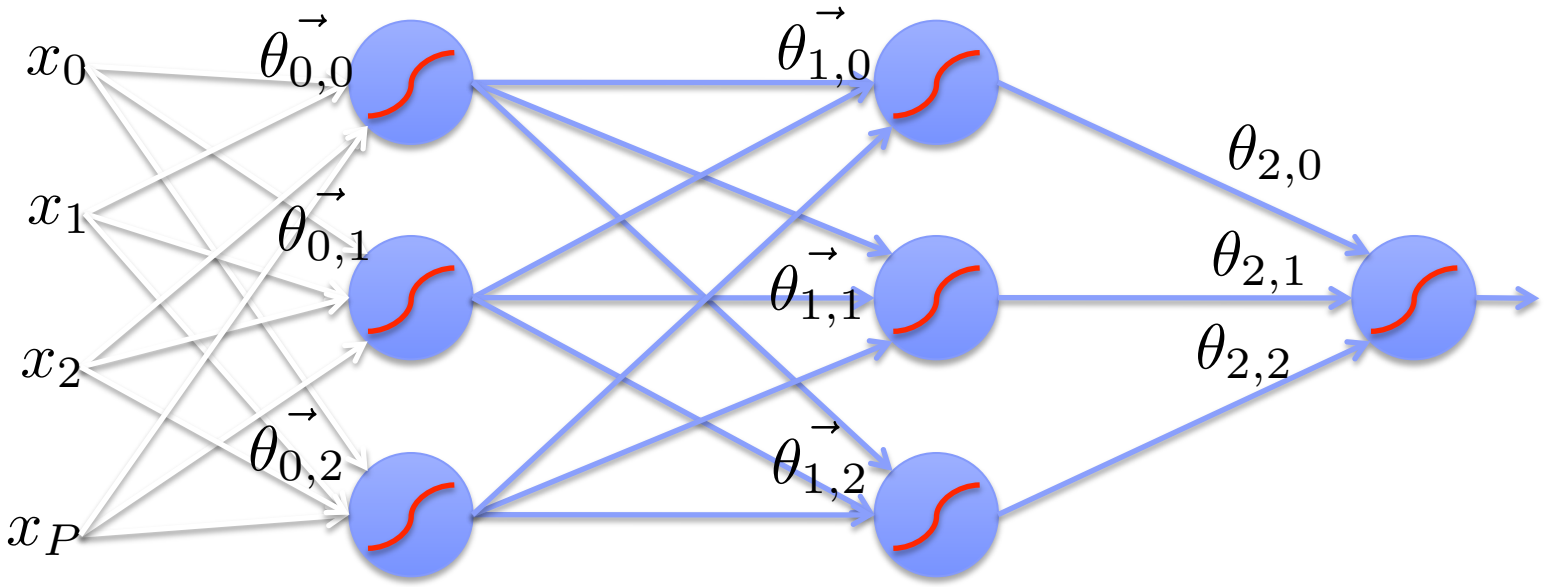
Training Neural Networks

- Most common approach to train neural networks is via stochastic gradient descent
  - Propagate input forward
  - Compute gradient of objective function
  - Propagate error gradient backwards



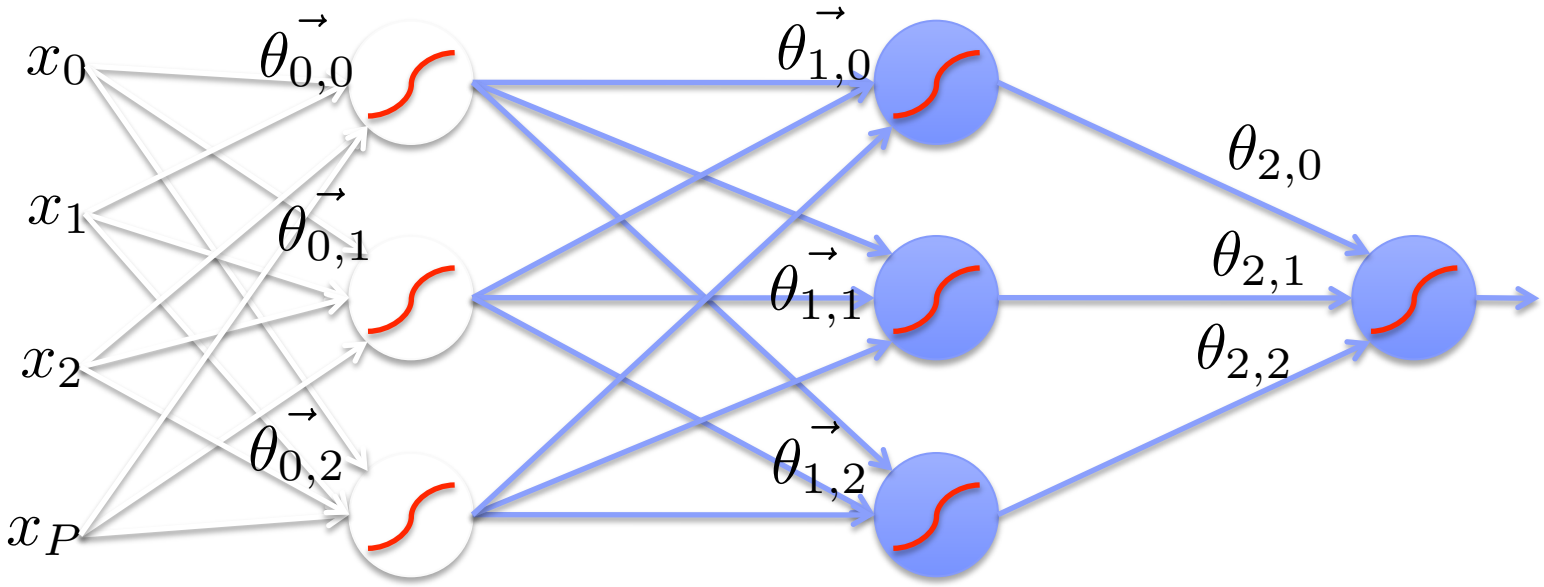
Feed-Forward Networks

- Predictions are fed forward through the network to classify



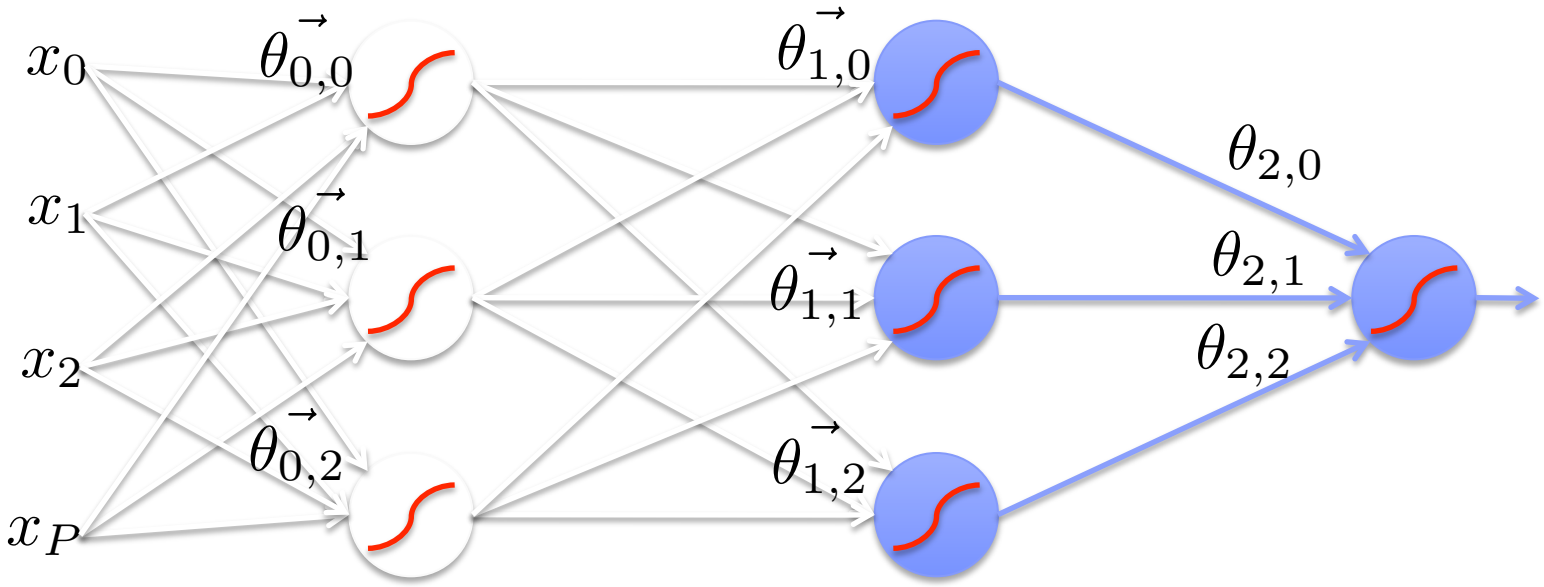
Feed-Forward Networks

- Predictions are fed forward through the network to classify



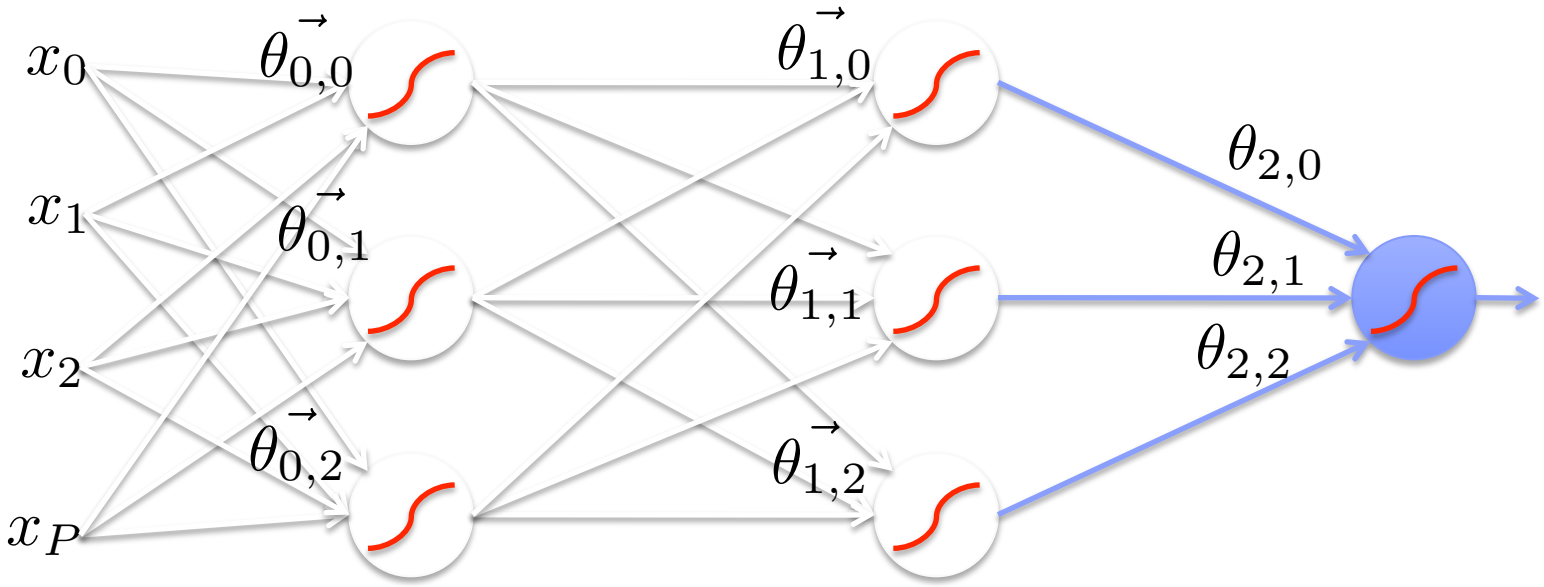
Feed-Forward Networks

- Predictions are fed forward through the network to classify



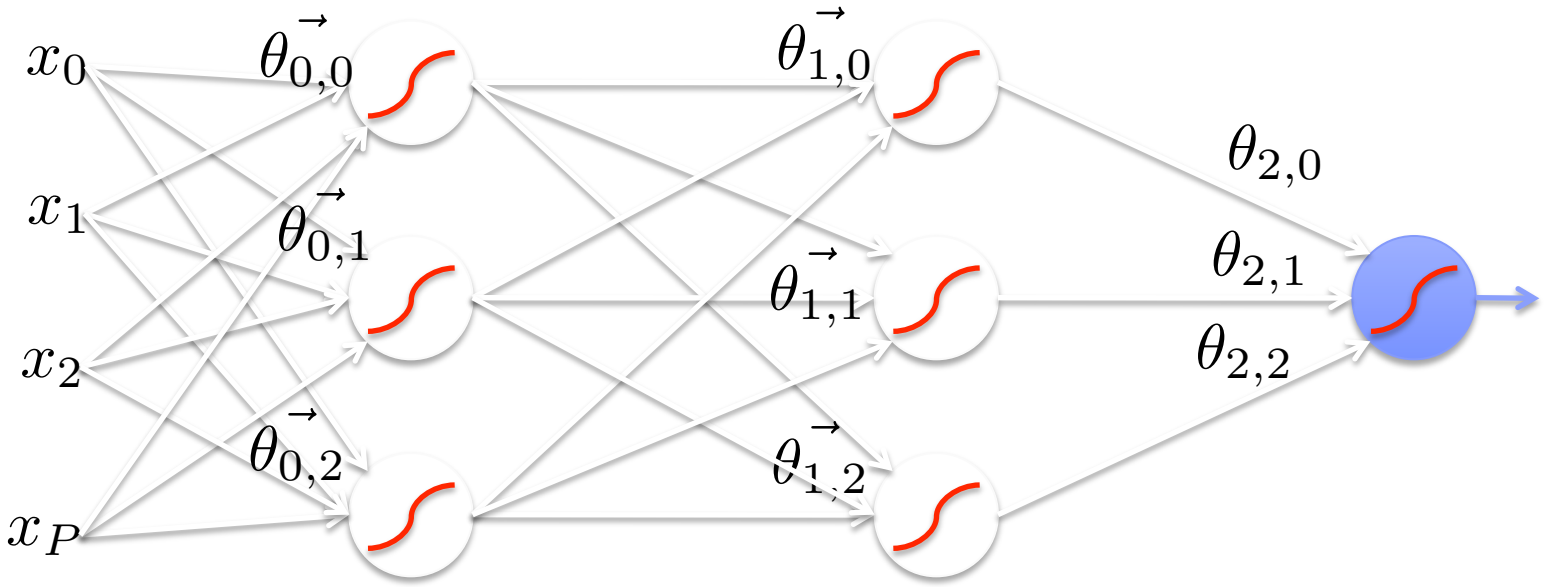
Feed-Forward Networks

- Predictions are fed forward through the network to classify



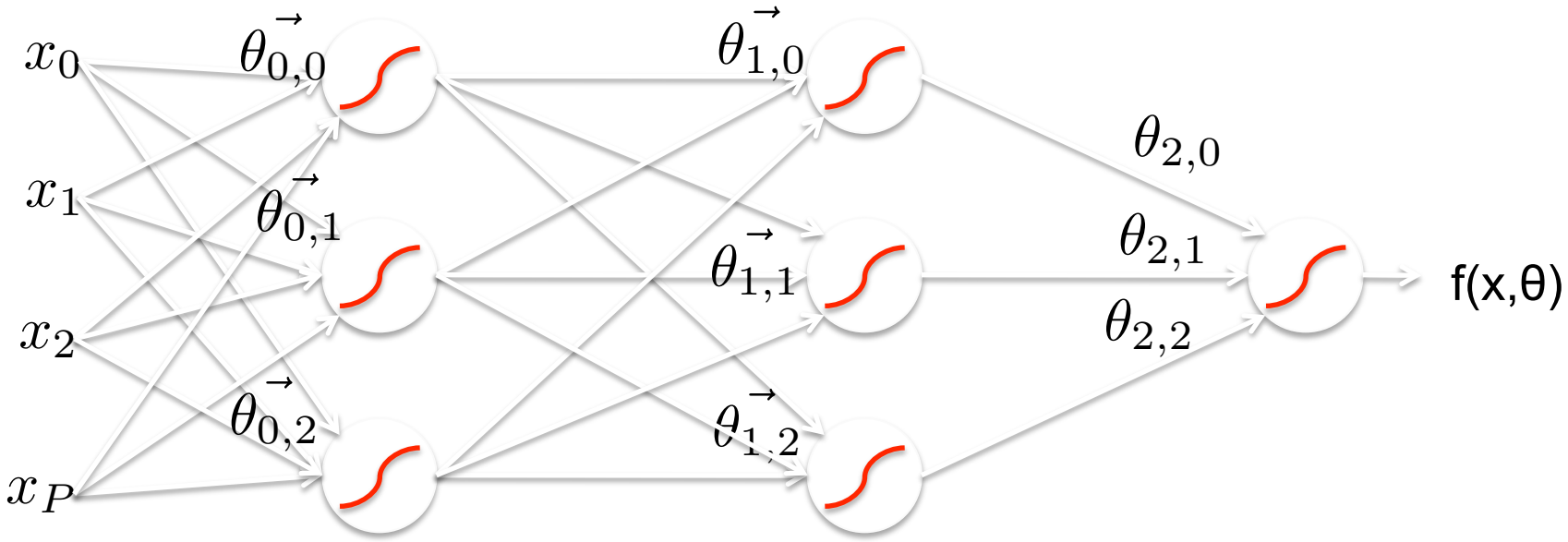
Feed-Forward Networks

- Predictions are fed forward through the network to classify

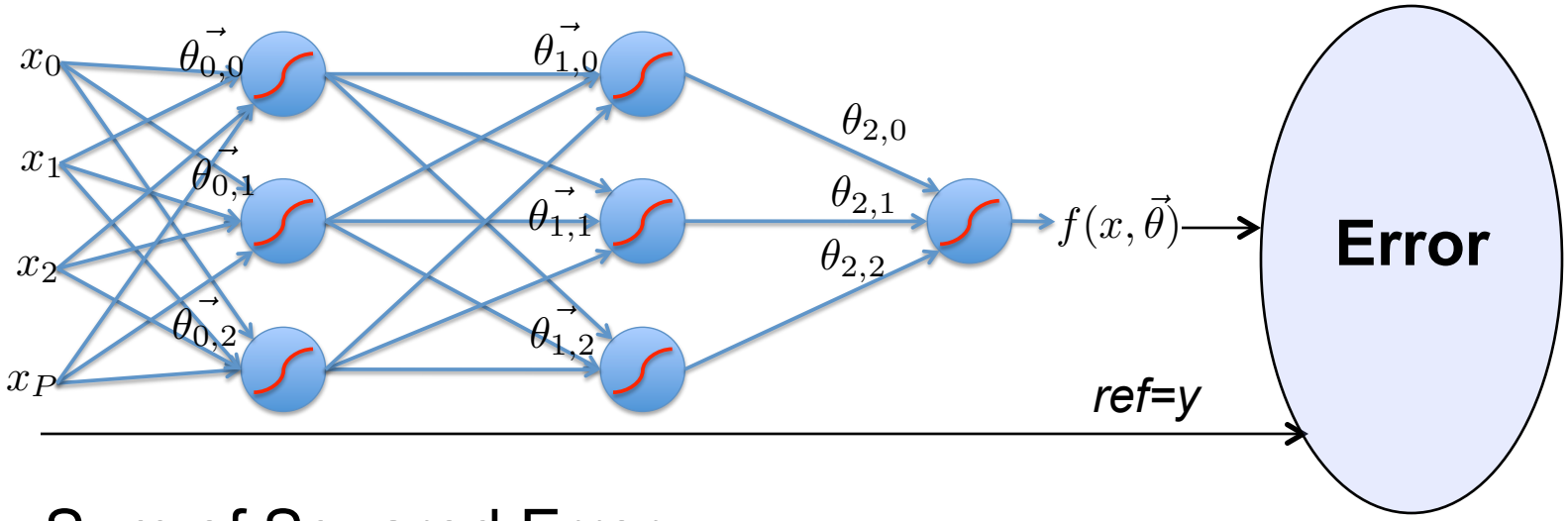


Feed-Forward Networks

- Predictions are fed forward through the network to classify



Define Objective Function



- Sum of Squared Error

$$y, f(x, \theta) \in R^N$$

$$L = \frac{1}{2}\sum_{n=1}^{N}(y_n - f_n(x, \theta))^2$$

- Cross-Entropy

$$y, f(x, \theta) \in [0,1]^N$$

$$\sum_{n=1}^{N} y_n = 1$$

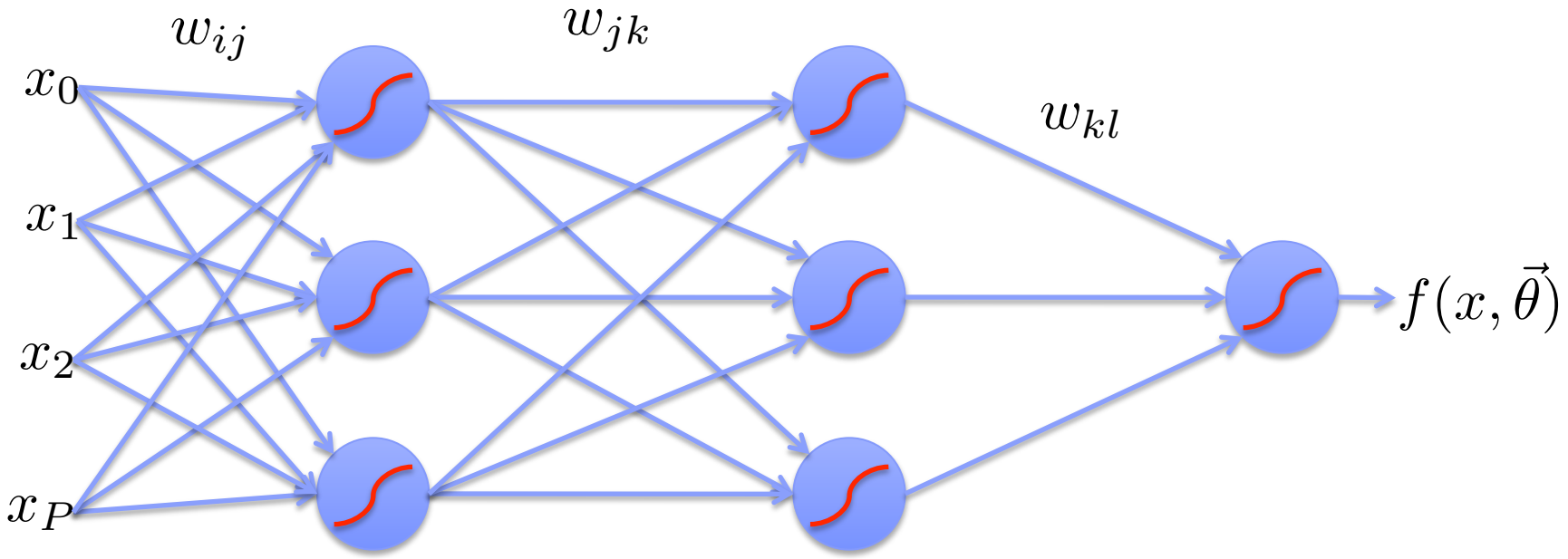
$$\sum_{n=1}^{N} f_n(x, \theta) = 1$$

$$L = -\sum_{n=1}^{N} y_n \log f_n(x, \theta)$$

Error Backpropagation

- Introduce variables over the neural network

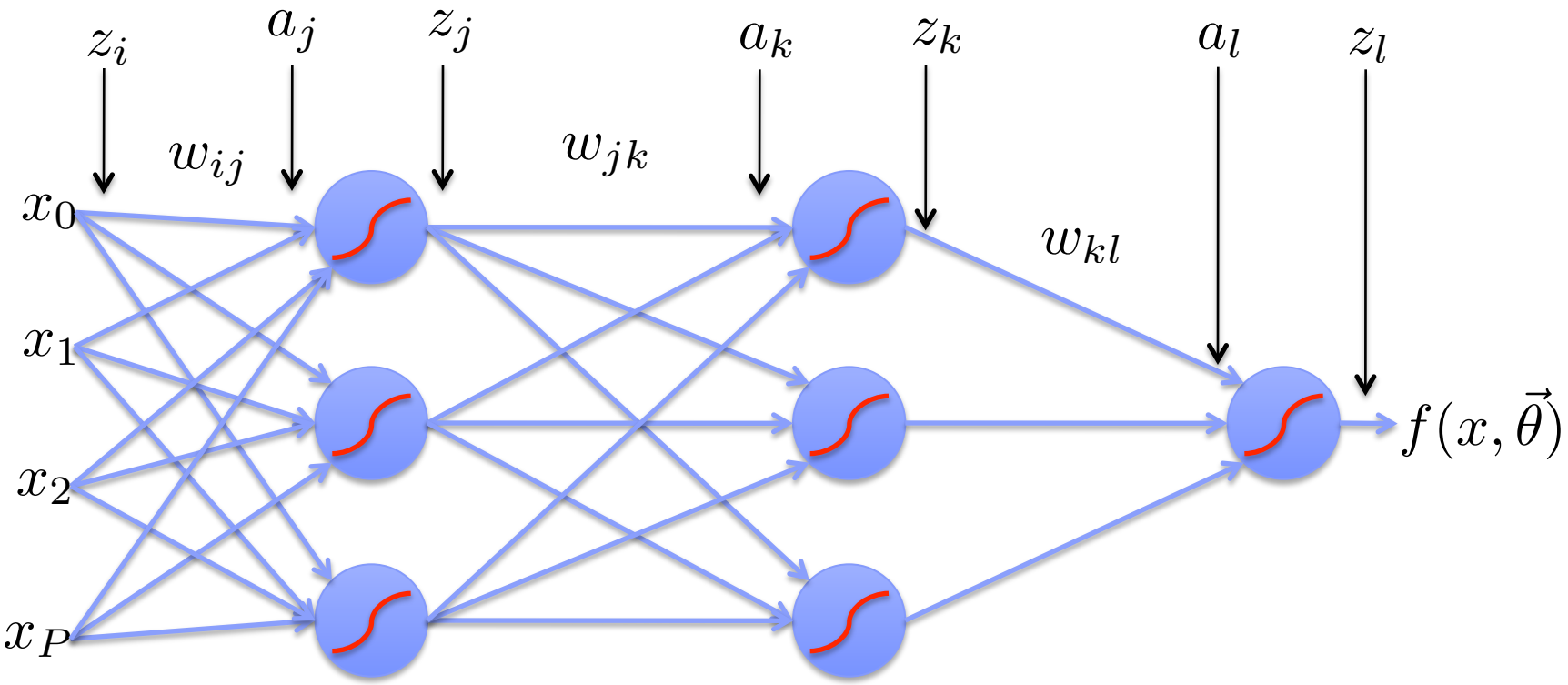
$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

- Introduce variables over the neural network
  - Define *a* to be the input of each non-linearity
  - Define *z* to be the output of each non-linearity

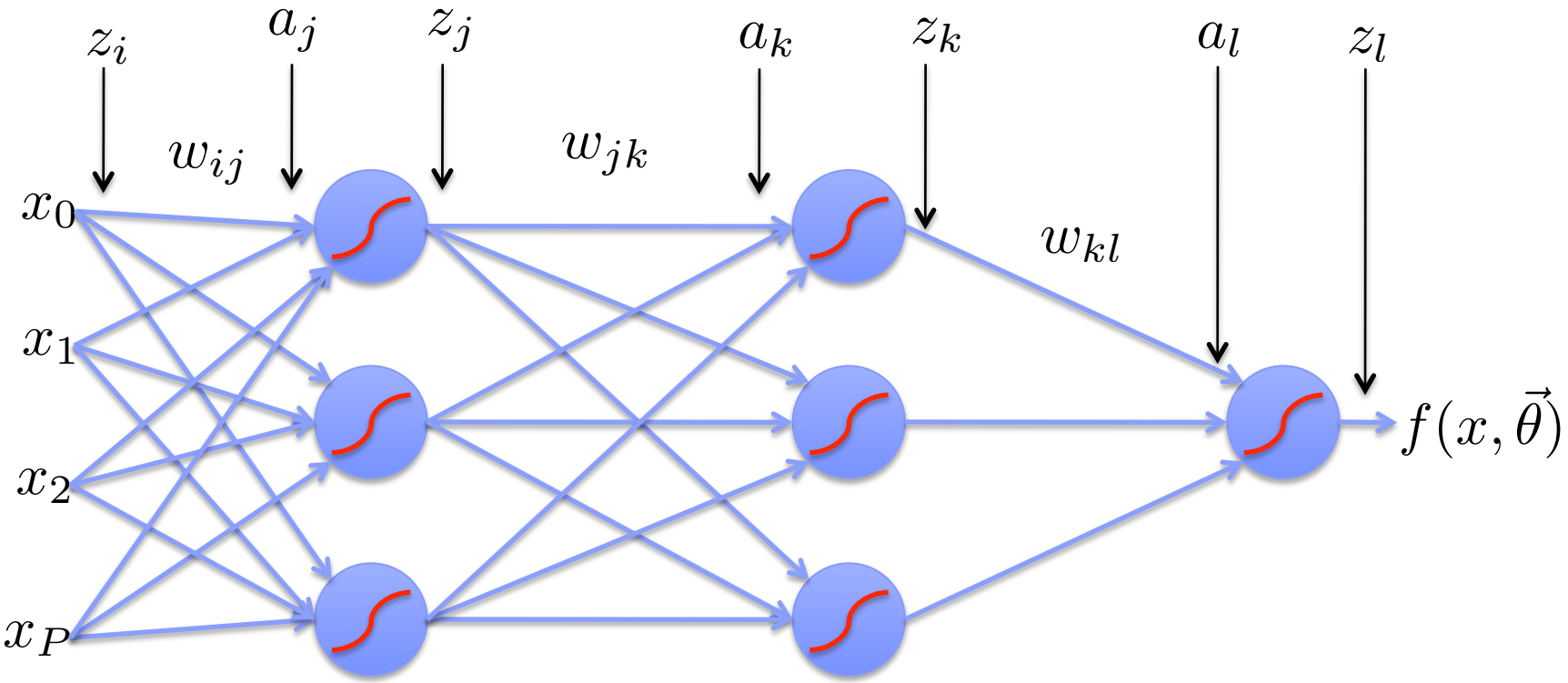


Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

$$a_j = \sum_i w_{ij} z_i \qquad a_k = \sum_j w_{jk} z_j \qquad a_l = \sum_k w_{kl} z_k$$

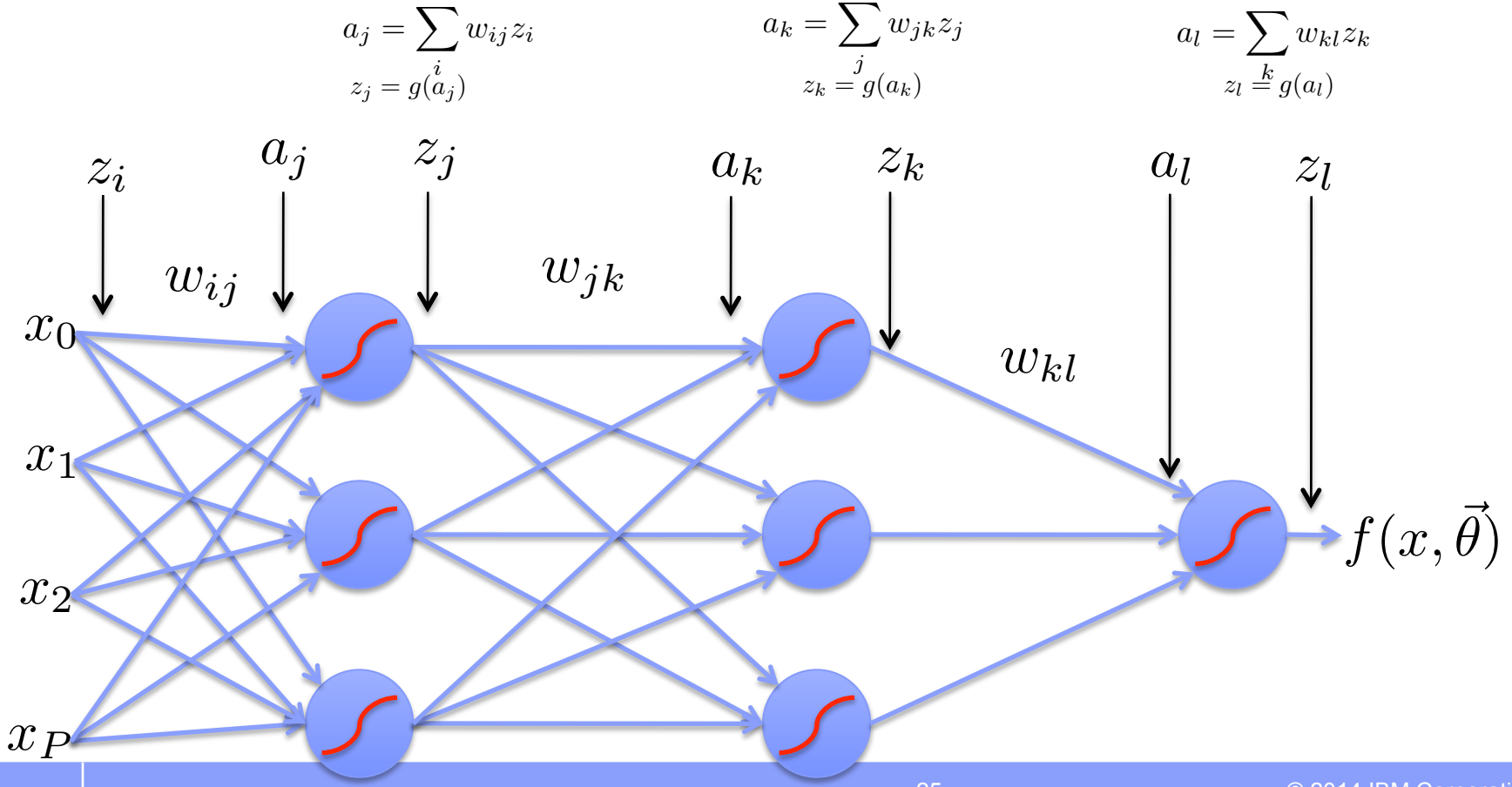
$$z_j = g(a_j) \qquad z_k = g(a_k) \qquad z_l = g(a_l)$$



Error Backpropagation

$$\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$$

Training: Take the gradient of the last component and iterate backwards



Error Backpropagation

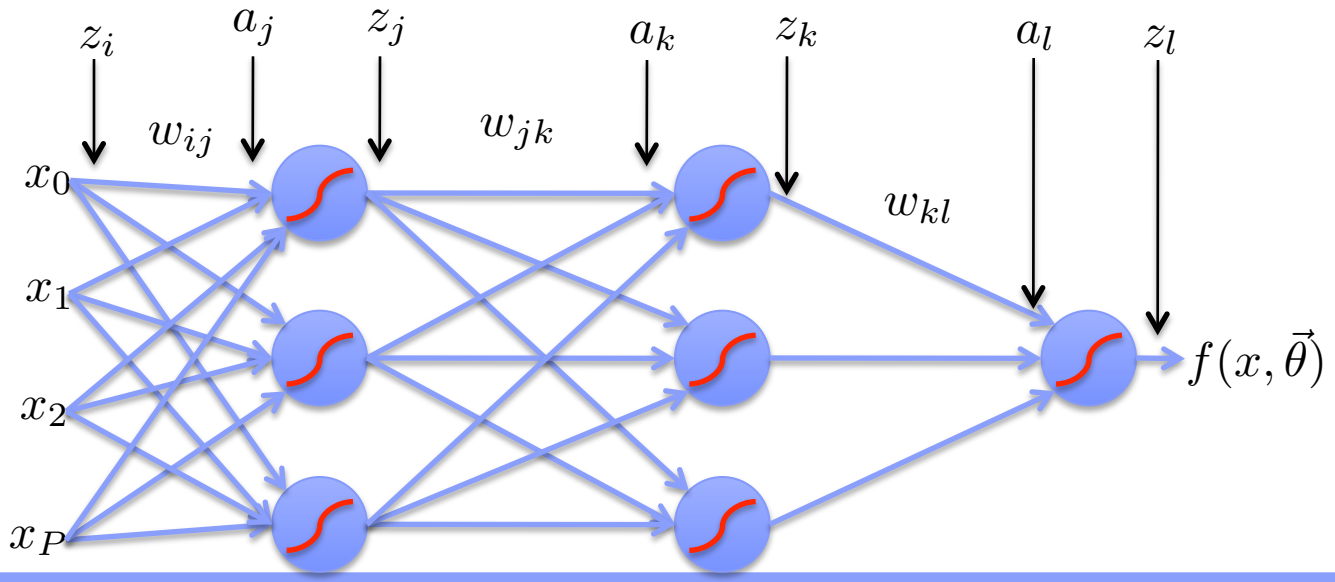
$$R(\theta) = \frac{1}{N}\sum_{n=1}^{N} L_n(y_n, f(x_n,\theta))$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{1}{2}(y_n - f(x_n,\theta))^2$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{1}{2}\left(y_n - g\left(\sum_k w_{k1}\, g\left(\sum_j w_{jk}\, g\left(\sum_i w_{ij} x_{n,i} \right) \right) \right) \right)^2$$

Empirical Risk Function

n is number of training points



Error Backpropagation

Optimize last layer weights w_{kl}

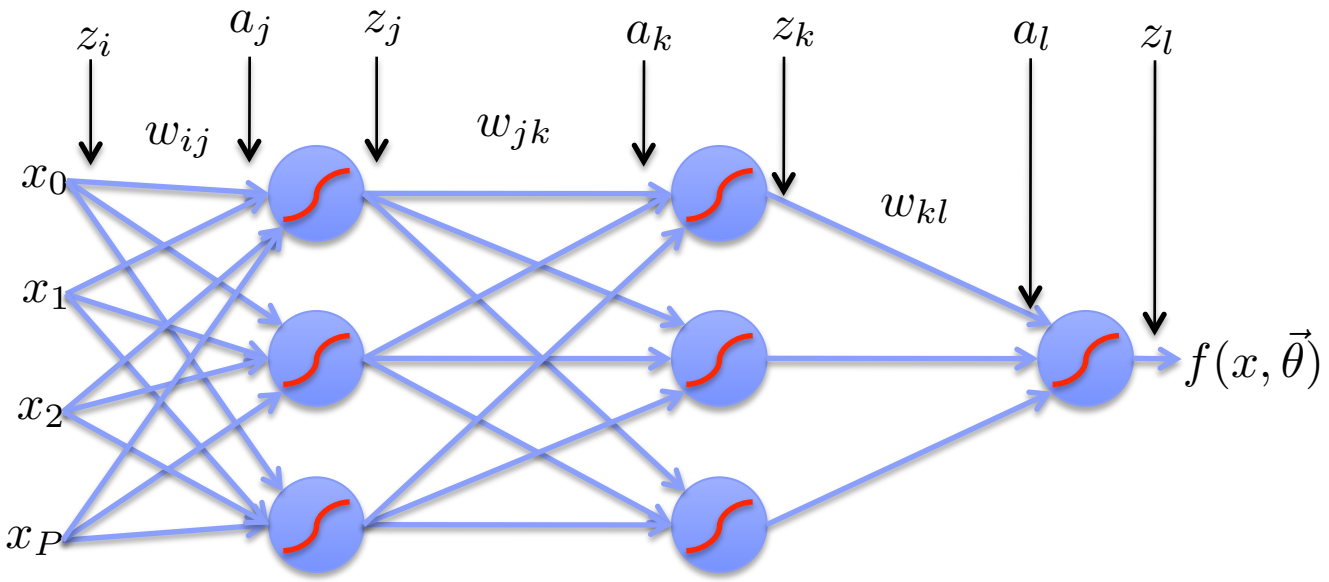
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$a_{l,n} = \sum_k w_{kl} z_k$$

$$f(x_n) = z_{l,n} = g(a_{l,n})$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule



Error Backpropagation

Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

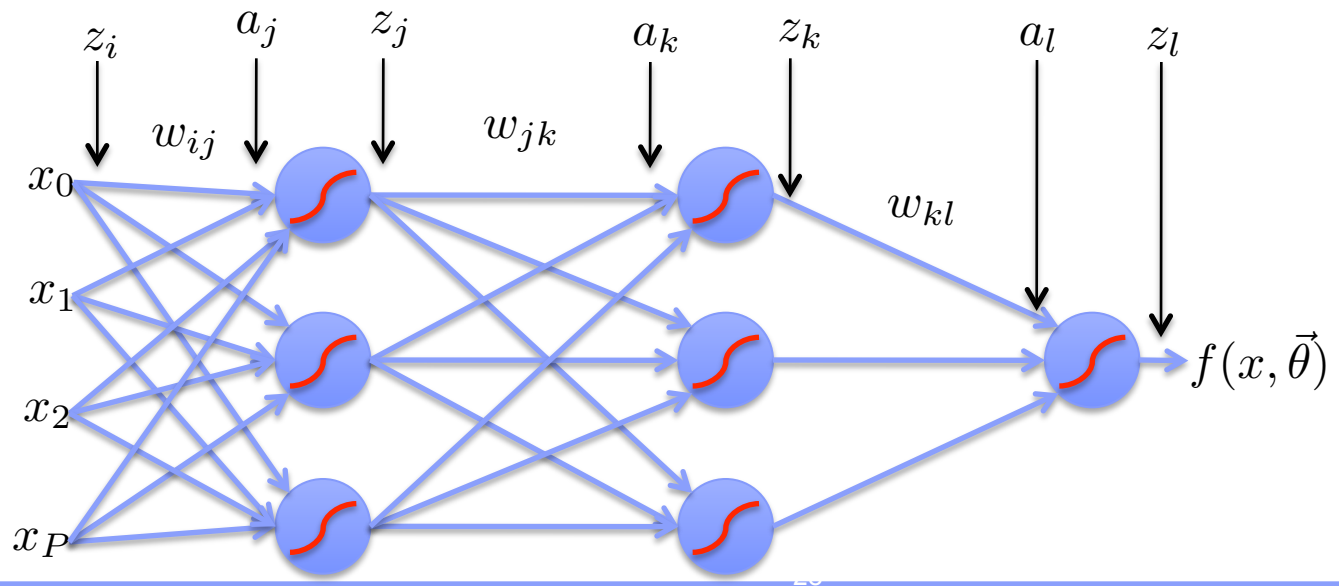
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}\left(y_n - g(a_{l,n})\right)^2}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

$$a_{l,n} = \sum_k w_{kl} z_k$$

$$f(x_n) = z_{l,n} = g(a_{l,n})$$



Error Backpropagation

Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

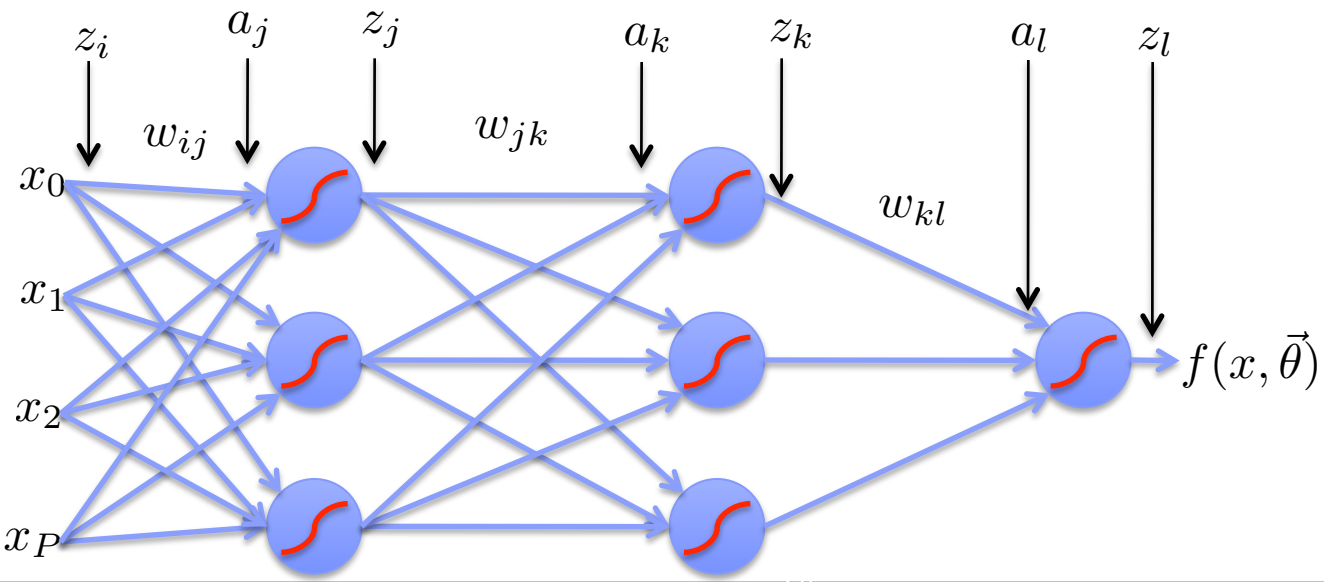
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}\left(y_n - g(a_{l,n})\right)^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right]$$

$$a_{l,n} = \sum_k w_{kl} z_k$$

$$f(x_n) = z_{l,n} = g(a_{l,n})$$



Error Backpropagation

Optimize last layer weights w_{kl}

$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

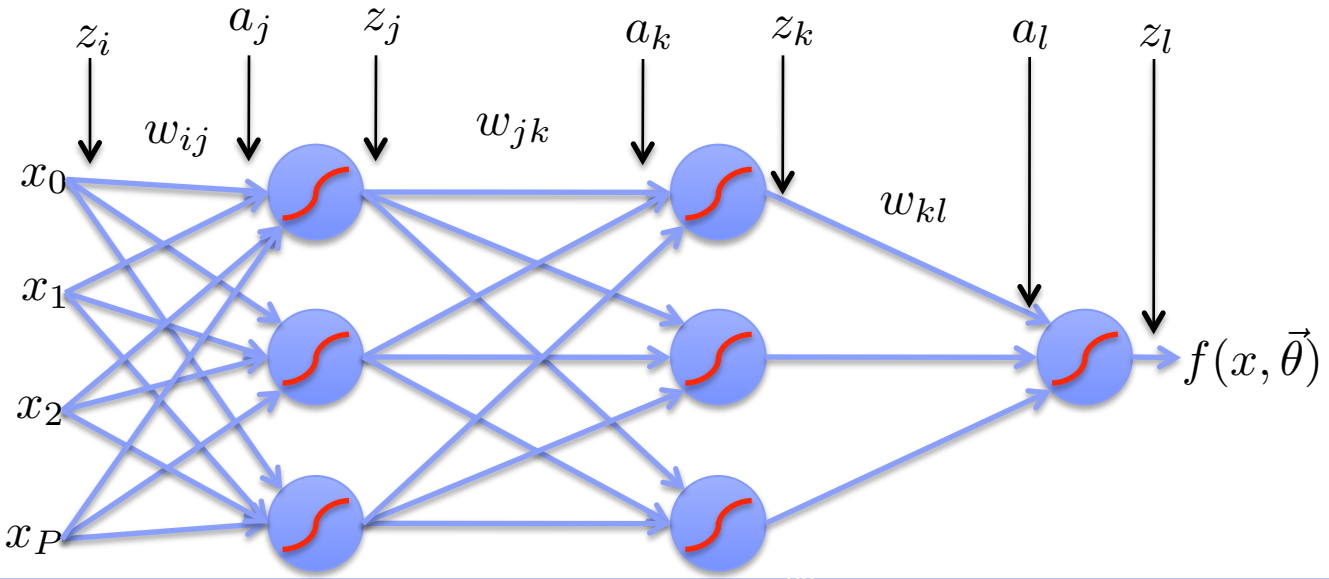
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$a_{l,n} = \sum_k w_{kl} z_k$$

$$f(x_n) = z_{l,n} = g(a_{l,n})$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$



Error Backpropagation

Optimize last layer weights w_{kl}

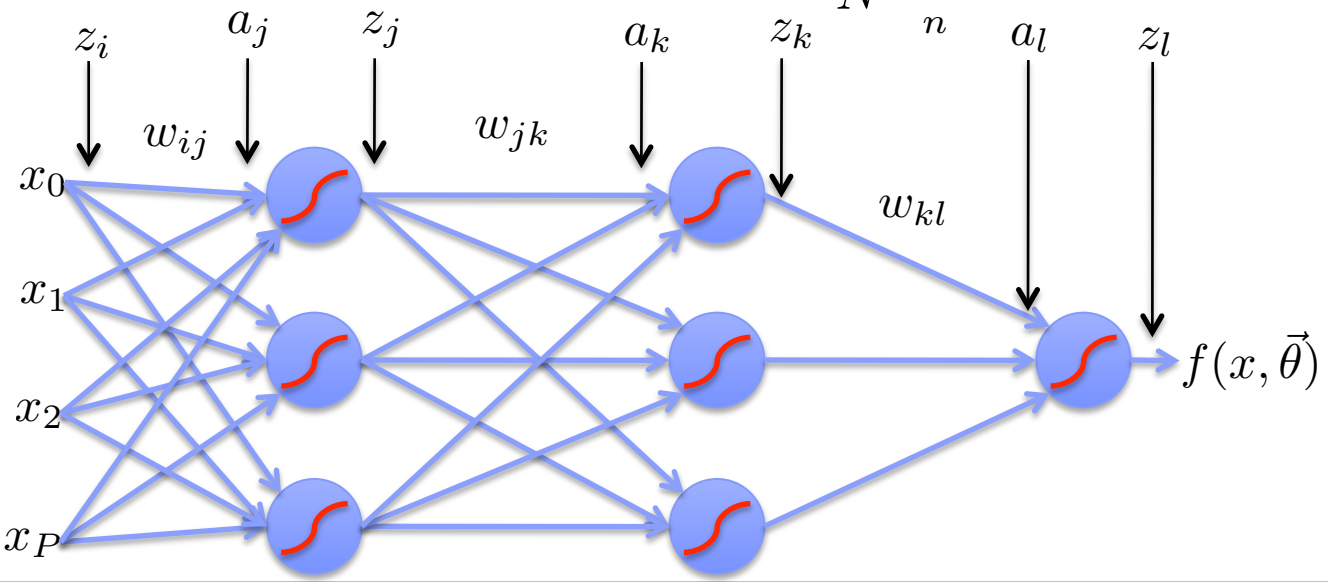
$$L_n = \frac{1}{2}\left(y_n - f(x_n)\right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right]$$

Calculus chain rule

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial \frac{1}{2}(y_n - g(a_{l,n}))^2}{\partial a_{l,n}}\right]\left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n}$$

$$= \frac{1}{N}\sum_n \delta_{l,n} \; z_{k,n}$$



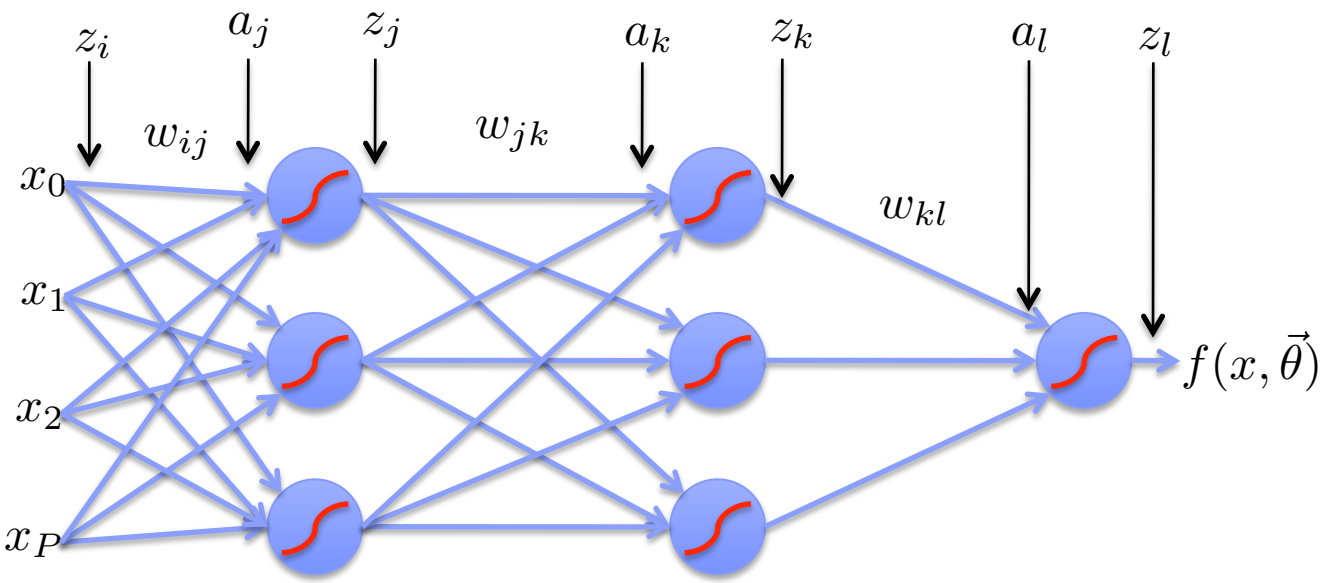
Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N} \sum_n \left[\frac{\partial L_n}{\partial a_{k,n}} \right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}} \right]$$

$$L_n = \frac{1}{2} \left(y_n - f(x_n) \right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \delta_{l,n} z_{k,n}$$



Error Backpropagation

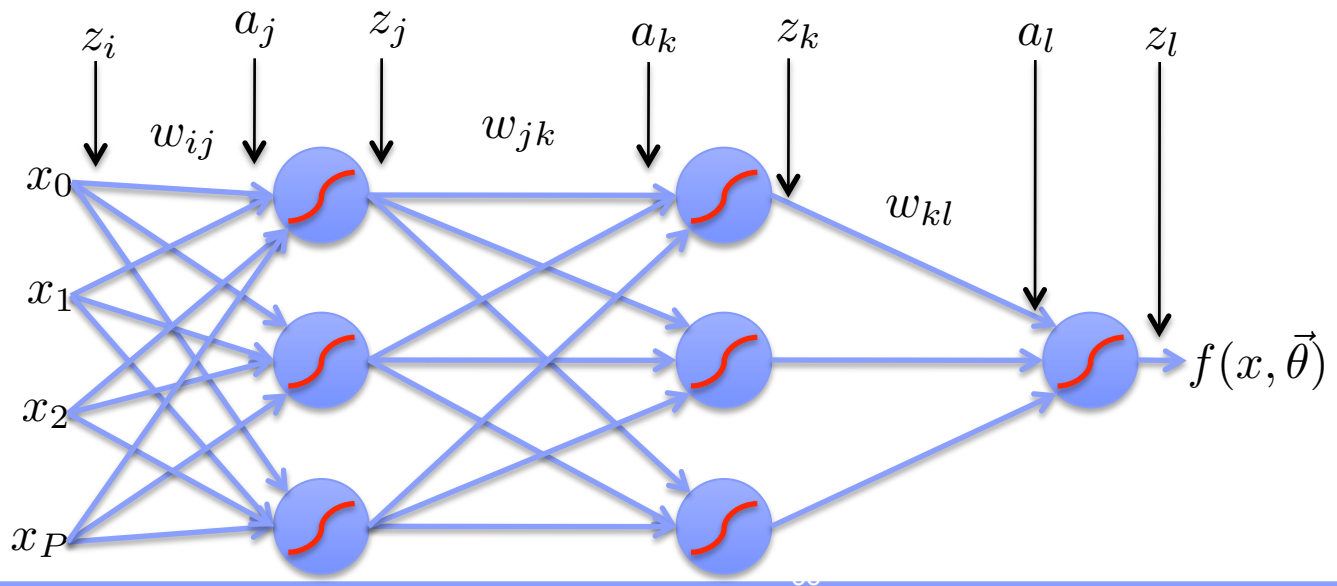
Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N} \sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}} \right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}} \right]$$

$$L_n = \frac{1}{2} \left(y_n - f(x_n) \right)^2$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \delta_{l,n} z_{k,n}$$

Multivariate chain rule



Error Backpropagation

Optimize last hidden weights w_{jk}

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}}\frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

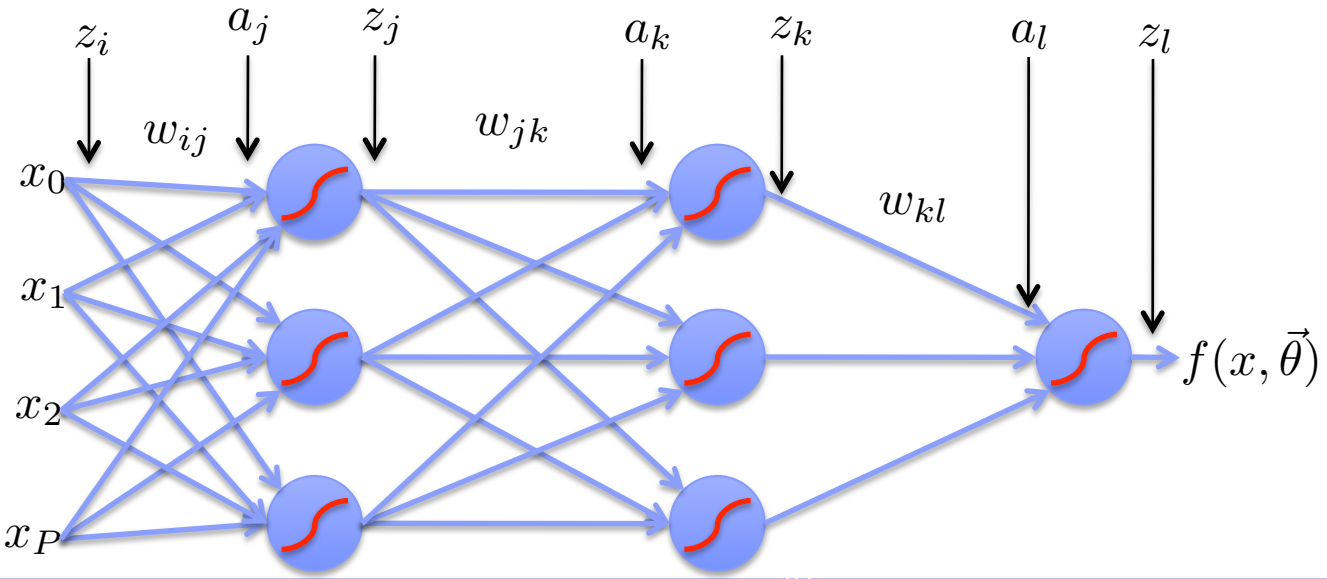
$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n}\frac{\partial a_{l,n}}{\partial a_{k,n}}\right]\left[z_{j,n}\right]$$

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

Multivariate chain rule

$$\frac{\partial L_n}{\partial a_{l,n}} = \delta_{l,n}$$

$$a_{k,n} = \sum_j w_{jk} z_{j,n}$$



Error Backpropagation

Optimize last hidden weights w_{jk}

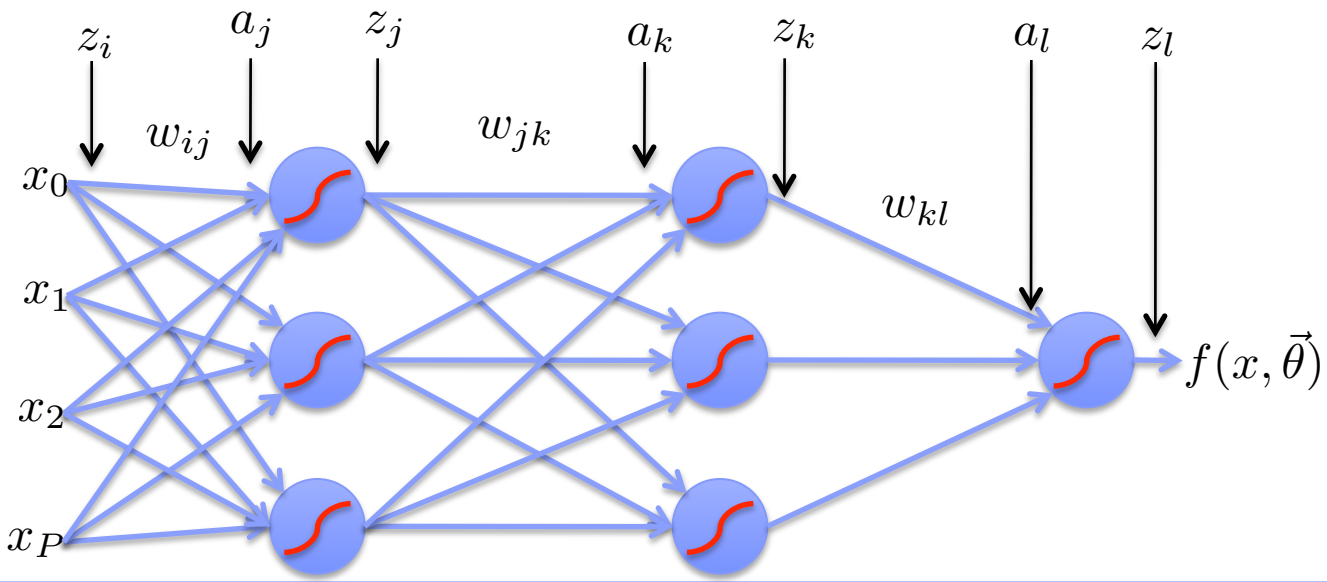
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[z_{j,n}\right]$$

$$a_{l,n} = \sum_k w_{kl} g(a_{k,n})$$



Error Backpropagation

Optimize last hidden weights w_{jk}

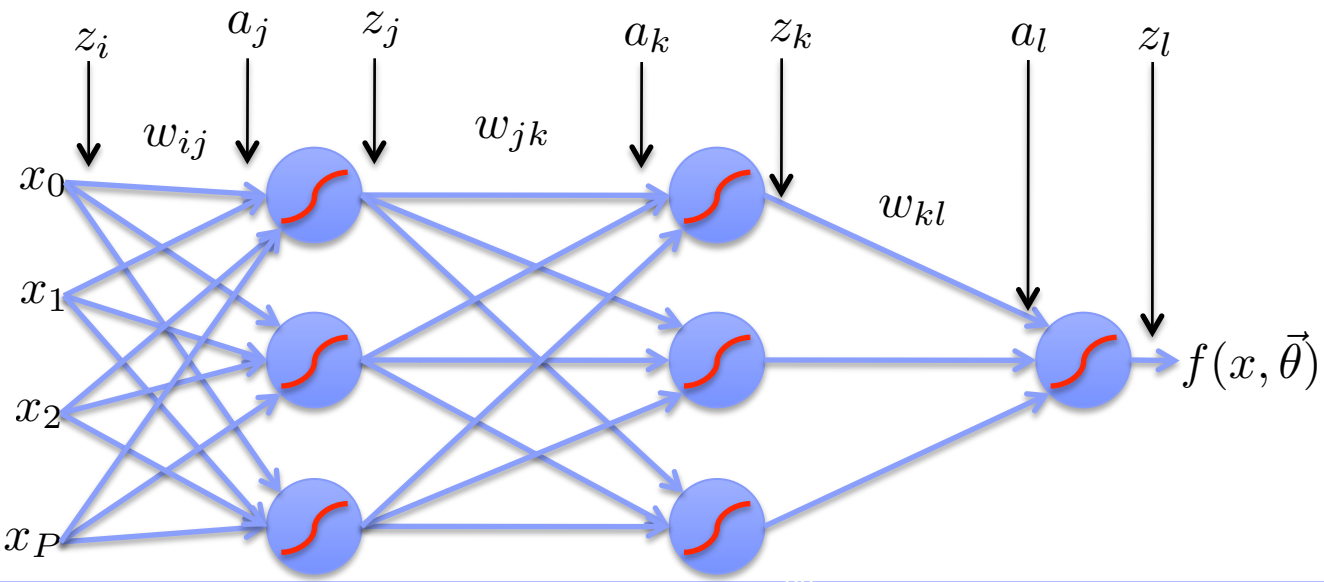
$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \frac{\partial L_n}{\partial a_{l,n}} \frac{\partial a_{l,n}}{\partial a_{k,n}}\right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right]$$

Multivariate chain rule

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\sum_l \delta_l w_{kl} g'(a_{k,n})\right] [z_{j,n}] = \frac{1}{N}\sum_n [\delta_{k,n}] [z_{j,n}]$$

$$a_{l,n} = \sum_k w_{kl} g(a_{k,n})$$



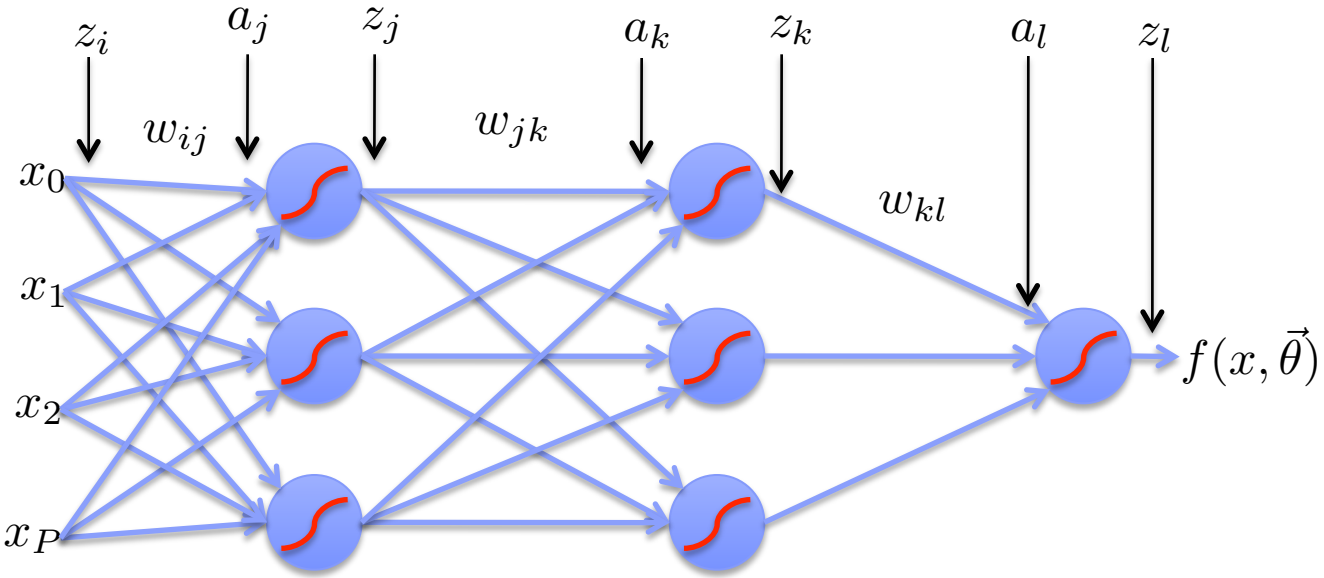
Error Backpropagation

Repeat for all previous layers

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{l,n}}\right]\left[\frac{\partial a_{l,n}}{\partial w_{kl}}\right] = \frac{1}{N}\sum_n \left[-(y_n - z_{l,n})g'(a_{l,n})\right] z_{k,n} = \frac{1}{N}\sum_n \delta_{l,n} z_{k,n}$$

$$\frac{\partial R}{\partial w_{jk}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{k,n}}\right]\left[\frac{\partial a_{k,n}}{\partial w_{jk}}\right] = \frac{1}{N}\sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n})\right] z_{j,n} = \frac{1}{N}\sum_n \delta_{k,n} z_{j,n}$$

$$\frac{\partial R}{\partial w_{ij}} = \frac{1}{N}\sum_n \left[\frac{\partial L_n}{\partial a_{j,n}}\right]\left[\frac{\partial a_{j,n}}{\partial w_{ij}}\right] = \frac{1}{N}\sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n})\right] z_{i,n} = \frac{1}{N}\sum_n \delta_{j,n} z_{i,n}$$



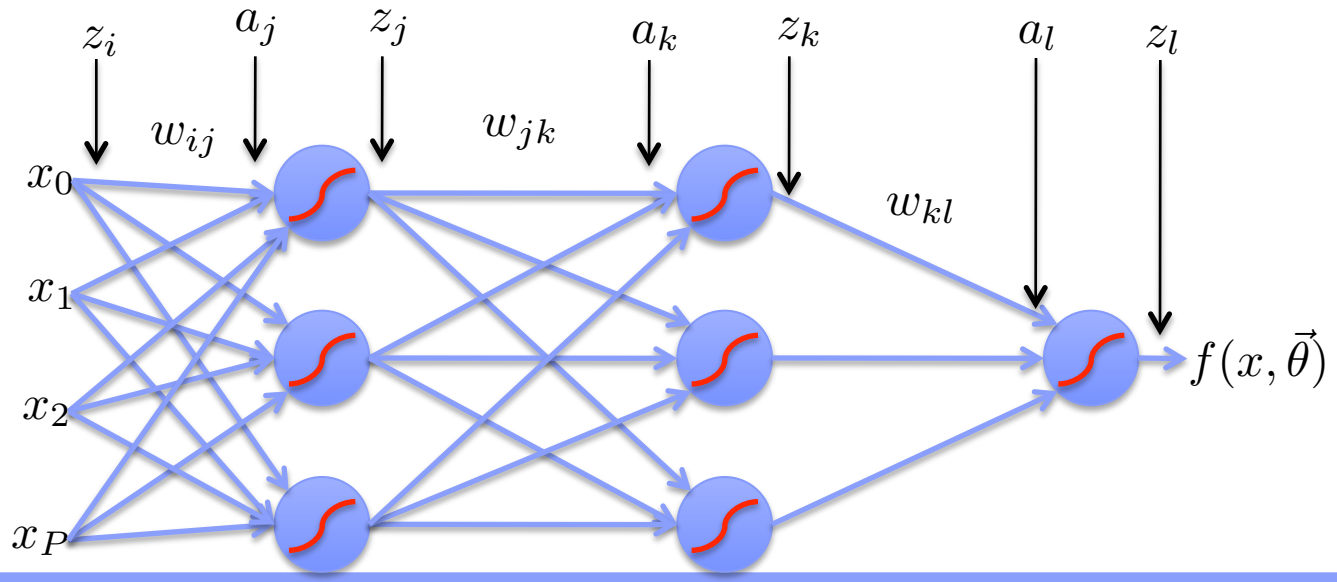
Error Backpropagation

Now that we have well defined gradients for each parameter, update using Gradient Descent

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \frac{\partial R}{w_{ij}}$$

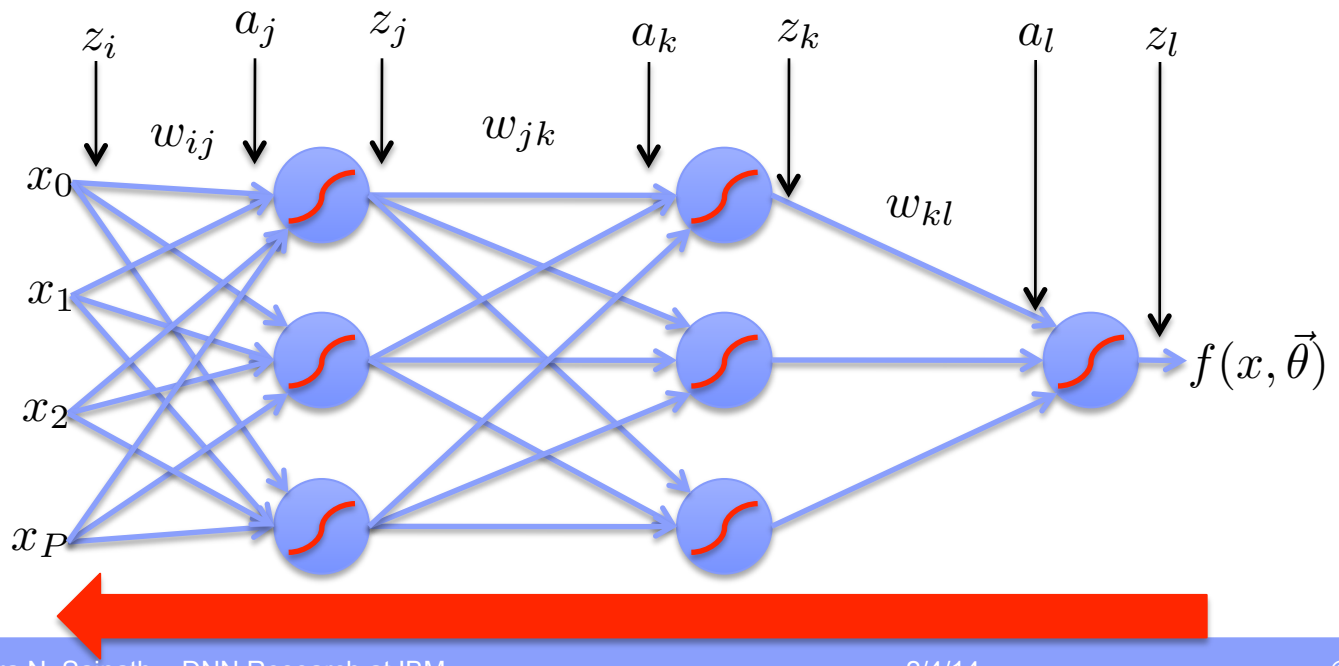
$$w_{jk}^{t+1} = w_{jk}^{t} - \eta \frac{\partial R}{w_{kl}}$$

$$w_{kl}^{t+1} = w_{kl}^{t} - \eta \frac{\partial R}{w_{kl}}$$



Error Back-propagation

- Error backprop unravels the multivariate chain rule and solves the gradient for each layer separately.
- The error δ is backpropagated along the network from one layer to the next



2nd Order Techniques

- SGD is not the only method to train NNs
- Training with SGD is not effective when networks become very deep
  - Objective function is non-convex and easier to get stuck in local minima
  - Areas of low-curvature become more likely
- 2nd order methods model this local curvature and can be much more effective

2nd Order Neural Network Training

- 1st order methods (SGD) update weights as follows, where θ is the weight, g is the gradient and η step size

$$\theta^{t+1} = \theta^{t} - \eta g^{t}$$

- 2nd order techniques update weights as follows, where H is the hessian

$$\theta^{t+1} = \theta^{t} - \eta (H^{t})^{-1} g^{t}$$

- Computing exact Hessian (Newton's method) is infeasible
- Popular 2nd order methods which do not explicitly compute the Hessian include Conjugate Gradient (CG) and L-BFGS

Outline

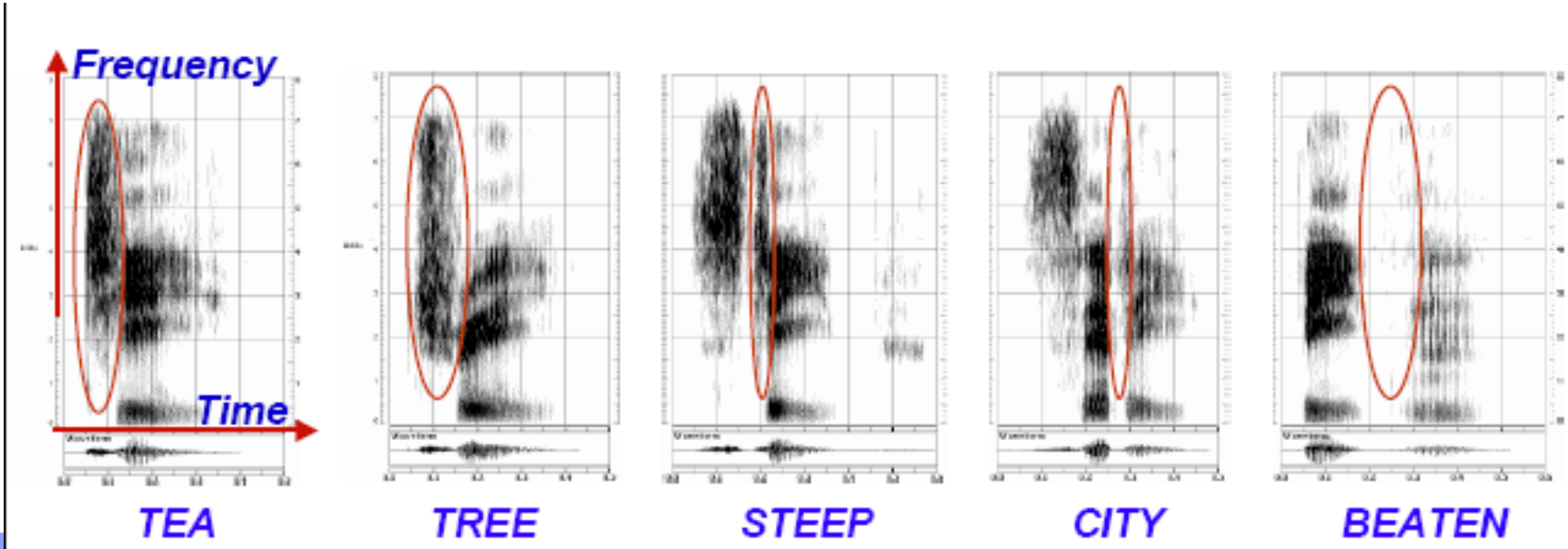
- Why Neural Networks?
- Training Neural Networks
- Making Deep Neural Network (DNNs) Successful For Speech Recognition
  - Pre-training
  - GPUs
- Research Directions and Challenges

Acoustic Modeling for Speech Recognition

- Speech recognition problem characterized as follows:

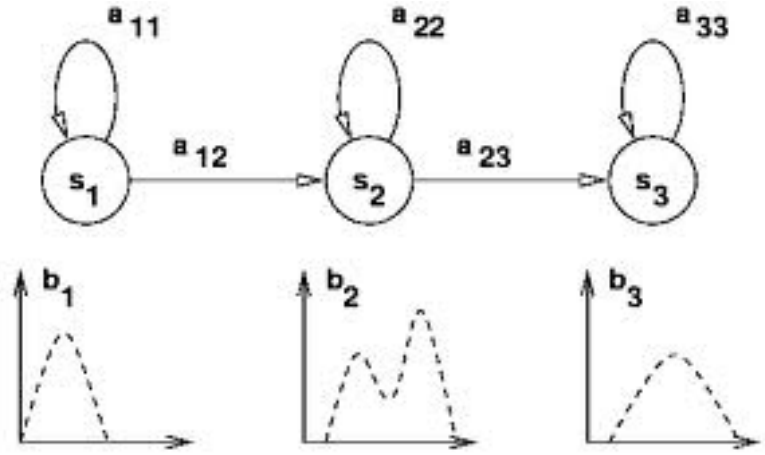
$$\widehat{W} = \arg\max_{W} P(W|A) = \arg\max_{W} P(A|W)P(W)$$

- Acoustic modeling is the process of modeling a set of sub-word units which make up words
- Acoustic realization of a phoneme depends strongly on context
- We model sub-word units as triphones (context-dependent states)



Acoustic Modeling for Speech Recognition

- Each sub-word unit is modeled by a 3-state Hidden Markov Model



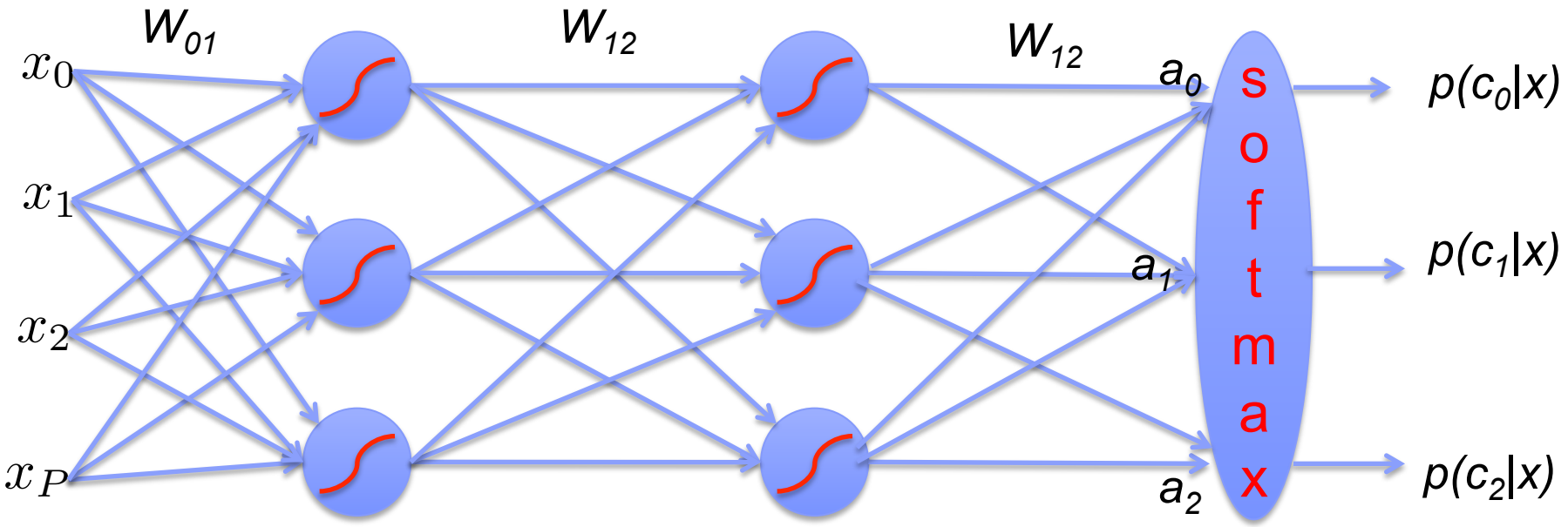
- The **most dominant** acoustic modeling technique thusfar is to model the output distribution in each state by a Gaussian Mixture Models (GMM)
- Neural Networks (alternatively called Multi Layer Perceptrons – MLPs) can also be used for acoustic modeling

Neural Network Acoustic Models

- Final non-linearity is represented by softmax

$$p(c_i|x) = \frac{\exp(-a_i)}{\sum_{j=1}^{n} \exp(-a_j)}$$

- Each class c_i will be the same sub-word units we build for GMMs
- One DNN is built to model all classes



Neural Network Acoustic Models

- Neural networks are trained to minimize cross-entropy objective function (i.e. frame error rate)

$$y_i^{ref}, p(c_i|x) \in [0,1]^N$$

$$\sum_{i=1}^{N} y_i^{ref} = 1$$

$$\sum_{i=1}^{N} p(c_i|x) = 1$$

$$L = -\sum_{i=1}^{N} y_i^{ref} \log p(c_i|x)$$

- NN gives posterior $p(c_i|x)$ so divide by class prior to get likelihood

$$p(x|c_i) = \frac{\mathrm{p}(c_i|x)}{p(c_i)}$$

- NN likelihood replaces GMM likelihood as output distribution in HMM (hybrid decoding)

Early Performance of Neural Networks

- Previous LVCSR performance with MLPs
  - shallow network (3 layers), small output targets (46) - [Zhu et al, ICSLP 2004]
  - On a Switchboard telephony task, gains with MLPs only observed when combined with baseline

Method	WER
Baseline GMM/HMM	30.8
+ MLP	28.6

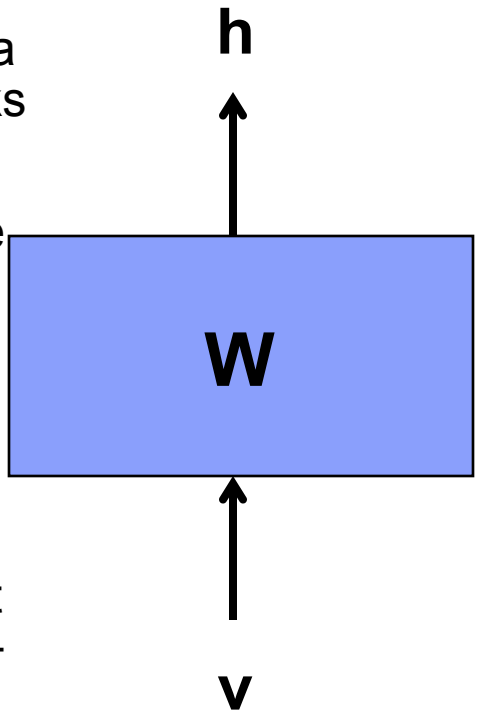
- Training neural networks is difficult!
  - Objective function is non-convex
  - Training is done SGD serially one one machine, can be slow on CPUs
- These difficulties pose challenges to have deep networks with many output targets

Making DNNs Successful for Acoustic Modeling

- 2 advances made DNNs successful for acoustic modeling
  1. Pre-training
  2. Improved Hardware with GPUs
- This encouraged
  - Deeper networks
  - Networks with more output targets (i.e., classes)

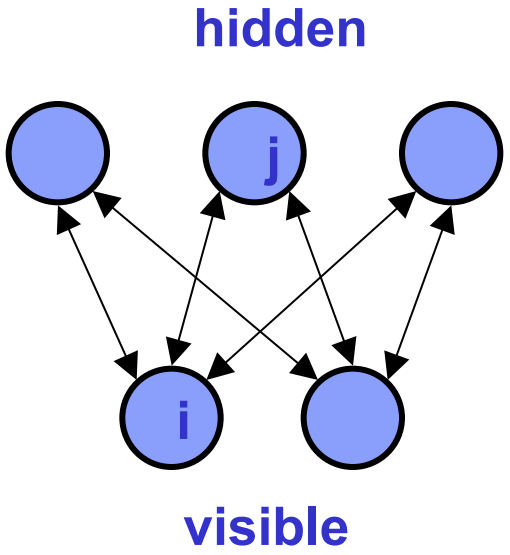
(1) Pre-Training via Unsupervised Learning

- The goal of unsupervised learning is to put the weights in a good initial space to encourage deeper and larger networks during fine-tuning
- Unsupervised learning systems can be designed using the encoder-decoder paradigm
  - Encoder: transform input *v* into code representation *h*
  - Decoder: reconstructs input from the code by minimizing reconstruction error
- Encoder-decoder paradigm learns weights such that the code captures higher-order relevant information from input signal, these weights are used to initialize network for fine-tuning
- Unsupervised learning
  - Restricted Boltzmann Machine (RBM) [Hinton – Toronto]
  - Sparse Encoding Symmetric Machine (SESM) [Lecun – NYU]
  - De-noising Auto-Encoder [Bengio – Montreal]



Restricted Boltzmann Machines

- Consider a one layer RBM
  - Weights are fully connected between hidden and visible units
  - No connections between hidden units
- Hidden units are conditionally independent given the visible states.
- Relationship between *v* and *h* for Bernoulli-Bernoulli RBMs given as:



$$p(h_j = 1 \mid \mathbf{v}) = 1/1 + \exp(-(\sum_i v_i w_{ij} + b_j)$$

$$p(v_i = 1 \mid \mathbf{h}) = 1/1 + \exp(-\sum_j h_j w_{ij} + b_i))$$

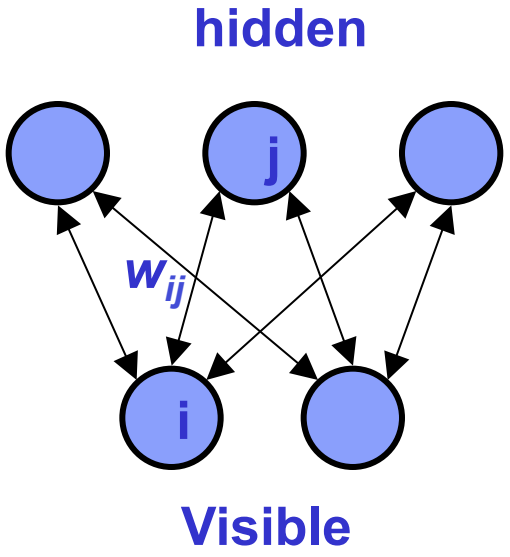
The Energy of Joint Configuration

- Each possible joint configuration of the visible and hidden units (ignoring biases) has an energy

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$

- The energy of a joint configuration of the visible and hidden units determines its probability:

$$p(v,h) \propto e^{-E(v,h)}$$



Using energies to define probabilities

- The probability of a joint configuration over both *v* and *h* depends on the energy *E* of that joint configuration compared with the energy of all other joint configurations.

$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}$$

partition function

- The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it.

$$p(v) = \sum_{h} p(v,h) = \frac{\sum_{h} e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}$$

How to maximize *p(v)*

- Goal of supervised fine-tuning is to maximize log *p(class|v)*
- It follows that the goal of unsupervised learning is to maximize *log p(v)*

$$w := w - \varepsilon \frac{\partial \log p(v)}{\partial w}$$

- Define free-energy as

$$F(v) = -\log \sum_h e^{-E(v,h)}$$

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$

$$p(v) = \frac{\sum_h e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}$$

- Gradient given as

$$-\frac{\partial \log p(v)}{\partial w} = \frac{\partial F(v)}{\partial w} - \sum_{\tilde{v}} p(\tilde{v}) \frac{\partial F(\tilde{v})}{\partial w}$$

Positive phase term

easy to compute directly

Negative phase term difficult

to analytically compute

Computing Derivative of Negative Phase

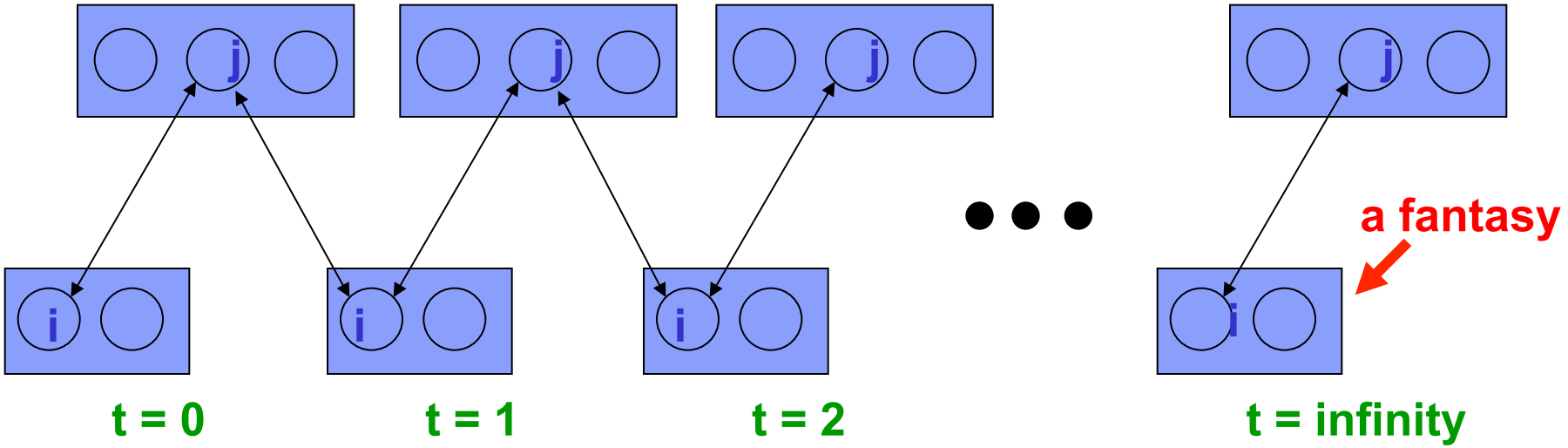
$$-\frac{\partial \log p(v)}{\partial w} = \frac{\partial F(v)}{\partial w} - \sum_{\tilde{v}} p(\tilde{v}) \frac{\partial F(\tilde{v})}{\partial w}$$

- Negative phase term can be represented as $E_p\left[\frac{\partial F(v)}{\partial w}\right]$, the expectation over all possible configurations of the input (under the distribution formed by the model), which is difficult to estimate analytically
- Estimate the expectation using a fixed number of model samples

$$-\frac{\partial \log p(v)}{\partial w} \approx \frac{\partial F(v)}{\partial w} - \frac{1}{|N|} \sum_{\tilde{v} \in N} \frac{\partial F(\tilde{v})}{\partial w}$$

- Obtain samples of *p(v)* using Gibbs sampling

Sampling in an RBM

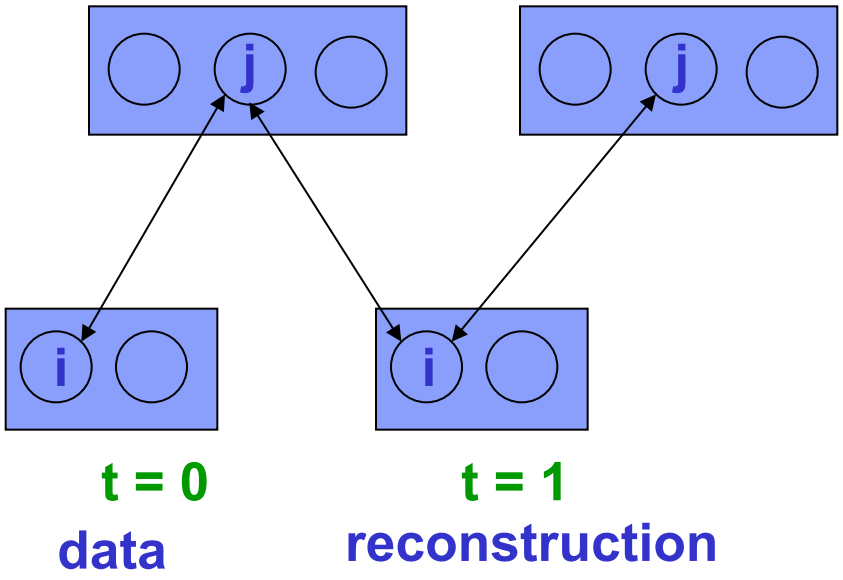


- In the RBM structure, *v* and *h* are conditionally independent
- To obtain samples of p(v,h):
  - Start with a training vector on the visible units.
  - Sample hidden units given fixed visible units
  - Then sample visible units given fixed hidden units

$$p(h_j = 1 \mid \mathbf{v}) = 1/1 + \exp(-(\sum_i v_i w_{ij} + b_j))$$

$$p(v_i = 1 \mid \mathbf{h}) = 1/1 + \exp(-\sum_j h_j w_{ij} + b_i))$$

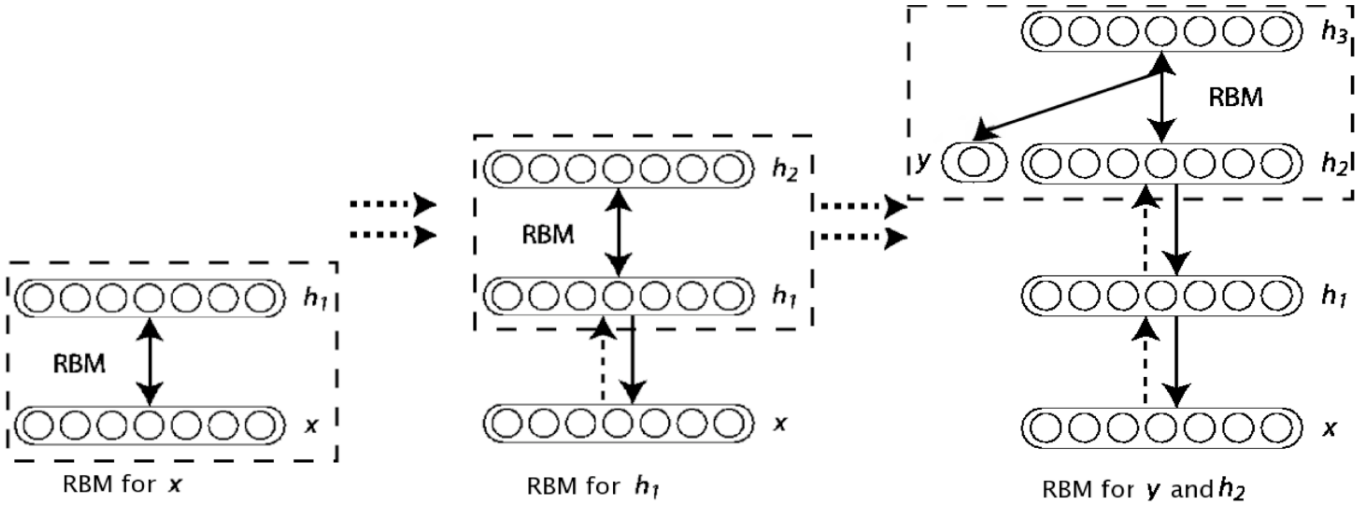
Contrastive Divergence



1. Start with a training vector on the visible units.
2. Update all the hidden units in parallel
3. Update the all the visible units in parallel to get a "reconstruction".
4. Update the hidden units again.

- With contrastive divergence, just one step of Gibbs sampling is run
- While this approximates -log p(v), it seems to work well in practice (Carreira-Perpinan & Hinton, 2005).

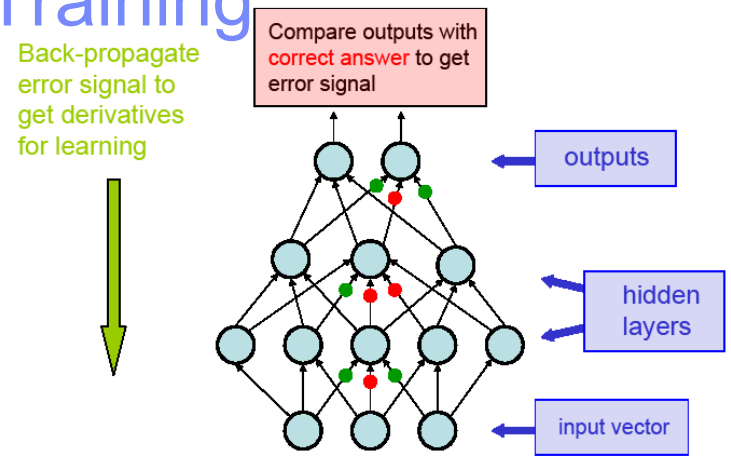
Constructing Deep Belief Networks



- First train a layer of features that receive input directly from the speech features
- Then treat the activations of the trained features as if they were speech features and learn features of features in a second hidden layer.
- Why greedy?
  - It can be proved that each time we add another layer of features we improve a variational lower bound on the log probability of the training data.
  - Simplicity of training

(2) Supervised Cross-Entropy Training

- Starting with pre-trained weights, retune weights via backpropagation using CE
- Most popular approach is mini-batch SGD
- Recipe is as follows [Morgan, 1995]:



1. For each training epoch
   2. Given all mini-batches $B=\{b_1, b_2, \ldots, b_N\}$, randomize batch order
   3. For each b_i in B, where $b_i=(v_1, \ldots, v_K)$
      4. Estimate gradient of loss function (i.e., cross-entropy)
      5. Update weights

$$w := w - \eta \sum_{k=1}^{K} \nabla_w f(w; v_k)$$

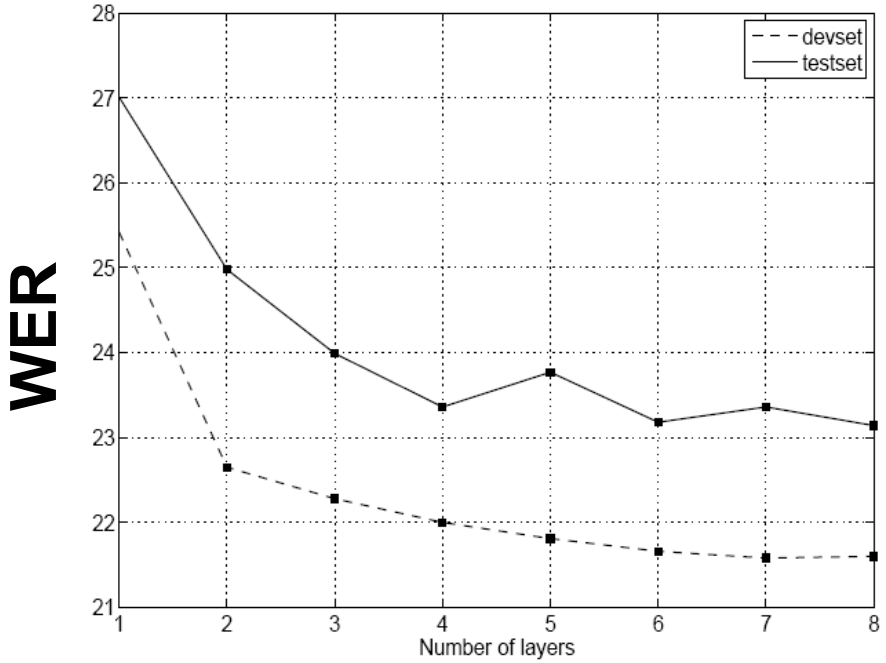
   6. Compute loss (objective function value) on a held-out set. Anneal weights if loss does not increase from previous iteration by more than 0.01
   7. Stop training once weights have been annealed 5 times

Analysis of Batch

- Utterance randomization:
  - Each batch is defined to be all frames from an utterance
  - Easier to implement since features can be read from file per utterance (like we do for Gaussian processing)
  - Because different batches look different (frames from different utterances), training is suboptimal
- Frame Randomization:
  - Randomize all frames in the training data, each batch is a random selection of frames
  - More difficult to implement since need to randomize all frames per epoch
  - Because different batches look more similar, training is more optimal
- Results indicate that frame randomization provides >5% WERR!

	Broadcast News (50 Hours)	Switchboard (300 Hours)	Broadcast News (400 Hours)
Utterance Randomization	18.5	17.8	17.2
Frame Randomization	17.8	16.1	16.5

Results with PT+CE FT: Deepness



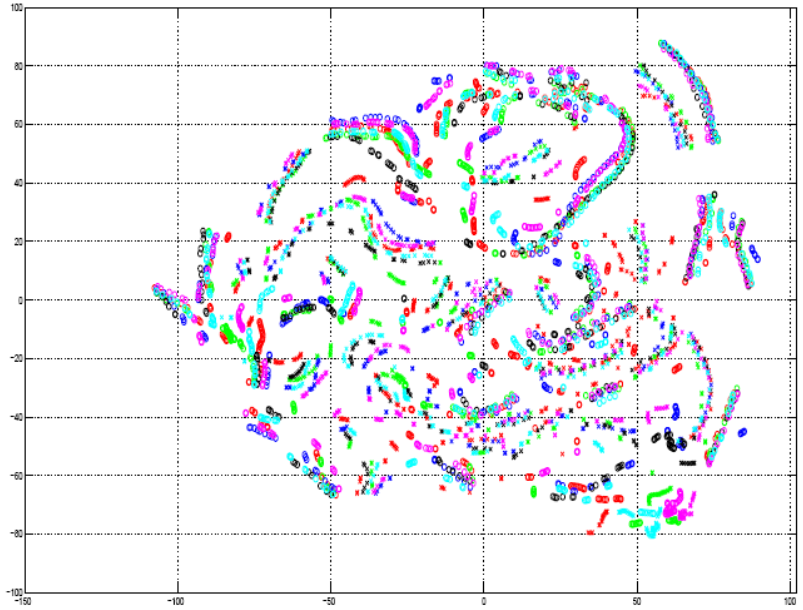
- Experimentally, we see that network depth improves WER
- Generally 6-7 hidden layers is used for speech tasks

Why Deepness in Speech?

[A. Mohamed at al, ICASSP 2012]

- t-SNE [van der Maaten, JMLR 2008] plots produce 2-D embeddings in which points that are close together in the high-dimensional vector space remain close in the 2-D space
- Analyze activations from 8 different speakers in TIMIT saying the same utterance
- Each color indicates a different speaker
- Similar **phones from different-speakers** are grouped together better at lower layers

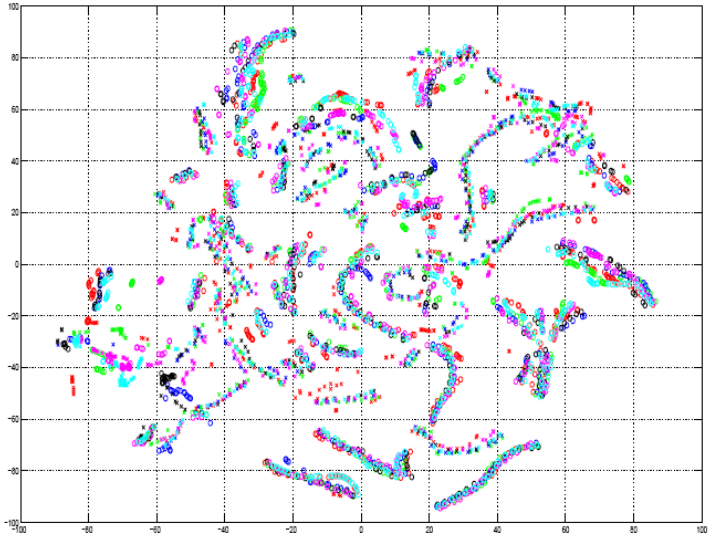
t-SNE plot, layer 1



Why Deepness in Speech?

- In higher layers, **better discrimination** between classes is performed
- These plots show us that DNNs are learning a speaker-adapted, discriminative feature representation jointly while doing the classification
- This makes DNNs very powerful compared to GMM/HMMs where features are designed separately from the classifier

t-SNE plot, layer 8



Results with PT+CE: Increased Output Targets

- We know with GMM/HMMs, increasing the number of context-dependent states (i.e., classes) improves performance
- In the past, MLPs typically trained with small number of outputs
  - increasing output targets becomes a harder optimization problem and does not always improve WER
  - increases parameters → increases training time
- With DBNs, **pre-training** putting weights in better space, and thus we can **increase output targets** effectively

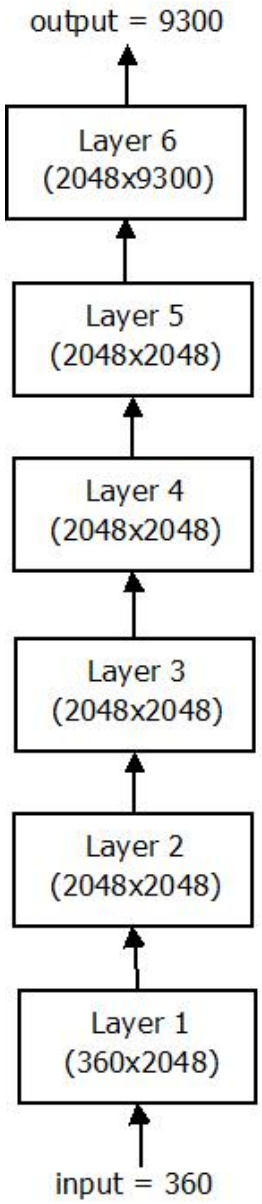
Number of Targets	WER
384	21.3
512	20.8
1024	19.4
2,220	**18.5**

Outline

- Why Neural Networks?
- Training Neural Networks
- Making Deep Neural Network (DNNs) Successful For Speech Recognition
  - Pre-training
  - GPUs
- Research Directions and Challenges

DNN Architecture for Speech Tasks

- We use the following architecture for speech tasks
  - 6-7 hidden layers
  - 1,024 – 2,048 hidden units
  - 4,000 – 9,000 output targets
- This amounts to anywhere between 8-40 million parameters!



GPU training

- GPUs divide complex computing tasks into thousands of smaller tasks that can be run concurrently
- One reason DNN training is slow is due to the large number of dense matrix multiplications
- GPUs help for SGD by parallelizing this matrix multiplication over thousands of cores

Features	Tesla K40	Tesla K20X	Tesla K20	Tesla K10
Number and Type of GPU	1 Kepler GK110B	1 Kepler GK110		2 Kepler GK104s
Peak double precision floating point performance	1.43 Tflops	1.31 Tflops	1.17 Tflops	0.19 Tflops
Peak single precision floating point performance	4.29 Tflops	3.95 Tflops	3.52 Tflops	4.58 Tflops
Memory bandwidth (ECC off)	288 GB/sec	250 GB/sec	208 GB/sec	320 GB/sec
Memory size (GDDR5)	12 GB	6 GB	5 GB	8 GB
CUDA cores	2880	2688	2496	2 x 1536

GPU training

- GPU training is the simplest approach to speed up SGD training
- We can achieve over a 5x speedup with a K10 GPU and a 9x speedup with a K20x speedup to a compared to a 8-core CPU for cross-entropy (CE) training
- But SGD does not work well for sequence training…
- GPUs can be expensive, what happens if we still want to use CPUs?

Method	WER (50-hr BN)	Training Time (hrs)
SGD (CPU)	17.8	35
SGD (GPU) – K10	17.8	7
SGD (GPU) – K20x	17.8	3.8

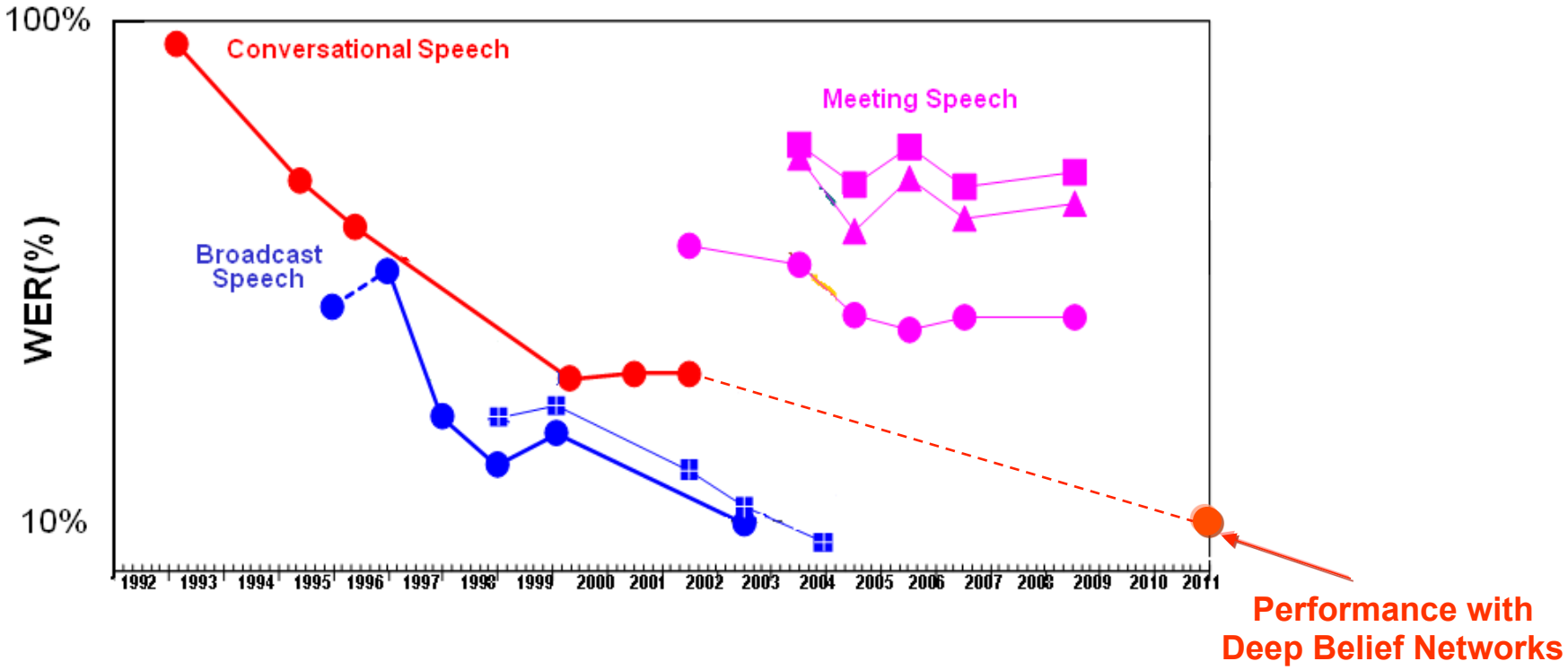
DNN Results @ IBM

- GMM/HMM system: speaker-adapted and discriminatively trained
- DNN: architecture
  - pre-trained, cross-entropy + sequence trained
  - 6 layers, 1024 or 2048 hidden units-layer, 2K-6K output targets
- DNNs provide between a 8-15% relative improvement in word error rate over GMM/HMM systems across a variety of tasks and languages
- Similar gains also reported by Microsoft and Google

	50 Hour Broadcast News	300 Hour Switchboard	400 Hour Broadcast News
GMM/HMM	17.2	14.3	16.5
DNN	14.9	12.2	15.2
% Relative Improvement	13.4	14.7	7.9

Historical Performance in Speech Recognition

- Few techniques we explore ***consistently*** show gains of this ***magnitude***

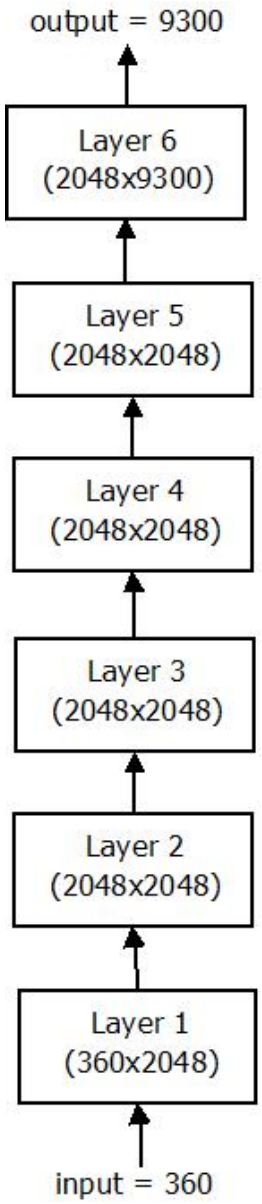


Outline

- Why Neural Networks?
- Training Neural Networks
- Making Deep Neural Network (DNNs) Successful For Speech Recognition
  - Pre-training
  - GPUs
- Research Directions and Challenges

DNN Architecture for Speech Tasks

- We use the following architecture for speech tasks
  - 6-7 hidden layers
  - 1,024 – 2,048 hidden units
  - 4,000 – 9,000 output targets
- This amounts to anywhere between 8-40 million parameters!
- Each weight update involves a large number of dense matrix multiplications:
  - propagating the input forward
  - computing the gradient
  - propagating the gradient backward



Challenges in Training Time

- While DNN performance in speech recognition is promising, our biggest overall challenge remains **training time**
  - 300 Hr SWB Timing Results (92 million frames)
    - Architecture - 360 dim input, 7 layers, 2,048 hidden units per layer, 9,300 output targets (43 million parameters)
    - On a 12 core CPU, SGD cross-entropy training takes 26.5 days!
- GMM systems are much quicker to train
    - 2 days to train same 300 hr SWB system
- Training is slow due to:
  - large number of **parameters**
  - large number of **dense matrix multiplications**
  - simplicity of **SGD** with serial training on one machine
- In reality, 300 hours is small for us. We want to train systems on 10,000+ hours plus of data

Novelties at IBM

- Performance Improvements:
  - Developed Sequence training [B. Kingsbury, ICASSP 2009] → 10% rel improvement
  - First to demonstrate success with Convolutional Neural Networks on LVCSR tasks [T. N. Sainath et al, ICASSP 2013] → 4-7% rel improvement over DNNs
  - Developed joint CNN/DNN Architecture [H. Soltau, G. Saon, T. N. Sainath, ICASSP 2014] → 5-10% rel improvement over CNNs alone

Novelties at IBM

- Improving Training Speed of DNNs:
  - 2nd order Hessian-free algorithm [B. Kingsbury et al, Interspeech 2012]
  - Low-rank Matrix Factorization [T.N. Sainath et al, ICASSP 2013]
  - HF training on Blue Gene [I. Chung et al, submitted to HPC 2014]

Conclusions

- Pre-training and GPUs encourage deeper networks and increased output targets
- DNNs can achieve between a 8-15% relative improvement compared to a strong GMM/HMM system across a wide variety of LVCSR tasks
- Further improvements can be achieved with CNNs, Sequence training
- Training time remains a huge challenge

References

- **DNNs for Acoustic Modeling**
  - G.E. Hinton, L. Deng, D. Yu, G. Dahl, A.Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," IEEE Signal Processing Magazine, 29, November 2012.
  - Abdel-rahman Mohamed, Geoffrey Hinton, Gerald Penn, "Understanding how Deep Belief Networks perform acoustic modelling", in ICASSP 2012.
  - N. Morgan and H. Bourlard, "Neural networks for statistical recognition of speech," *Proc. IEEE*, vol. 83, no. 5, pp. 742–772, May 1995.
  - F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proc. Interspeech*, 2011, pp. 437–440.
- **Pre-training**
  - G.E. Hinton, S. Osindero, Y. The,"A fast learning algorithm for deep belief nets," Neural Computation, 18, pp 1527-1554, 2006.
  - H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring Strategies for Training Deep Neural Networks," *Journal of Machine Learning Research*, vol. 1, pp. 1–40, 2009.
- **2nd order techniques/sequence training**
  - B. Kingsbury, T. N. Sainath, and H. Soltau, "Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Optimization," in Proc. Interspeech, September 2012.
  - J. Martens, "Deep learning via Hessian-free optimization," in *Proc. 27th Int. Conf. Machine learning*, 2010, pp. 735–742.
- **Alternative structures/applications of DNNs**
  - A. Graves, A. Mohamed, G. Hinton. Speech Recognition with Deep Recurrent Neural Networks. To appear in ICASSP 2013, Vancouver, Canada
  - T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and H, Černocký," RNNLM - Recurrent Neural Network Language Modeling Toolkit, in Proc. ASRU, 2011.
  - T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Auto-Encoder Bottleneck Features Using Deep Belief Networks," in Proc. ICASSP, March 2012.
  - T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep Convolutional Neural Networks for LVCSR," to appear in Proc. ICASSP, May 2013.

